ไมโครคอนโทรลเลอร์

Arduino

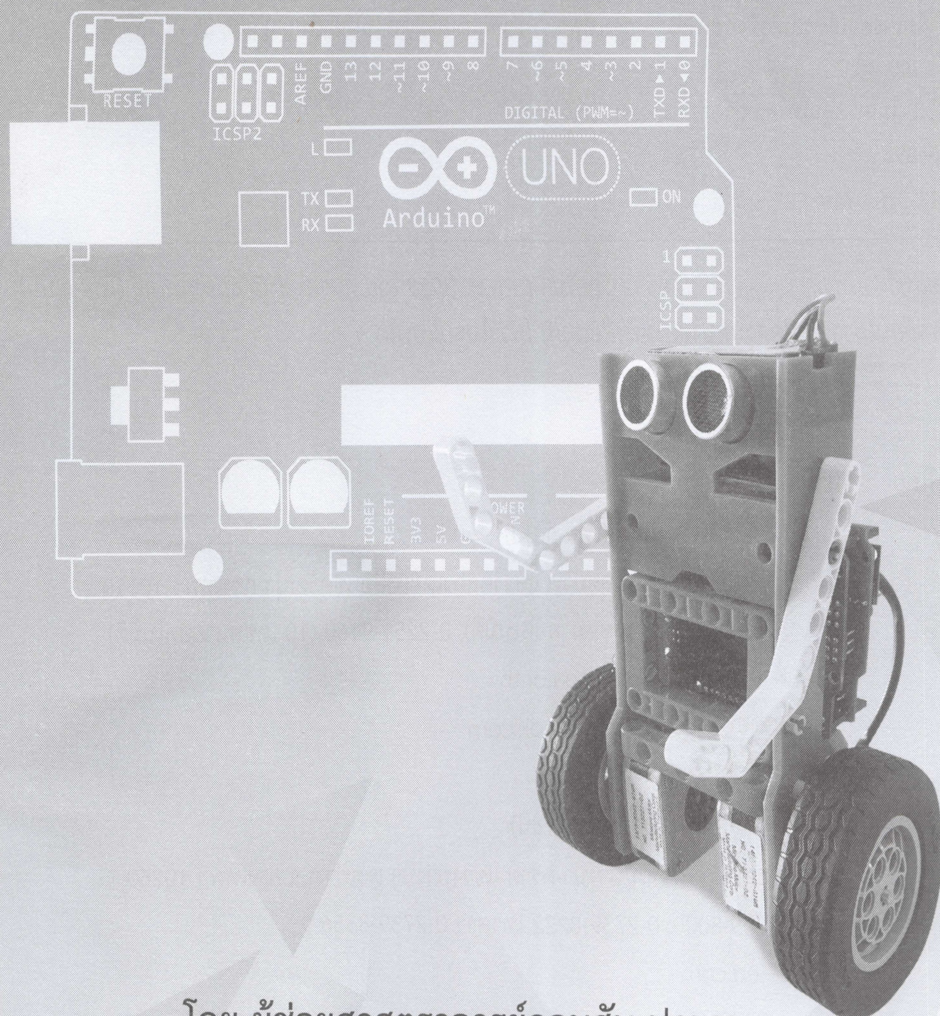
Arduino | Programming | Sensor | I/O | LCD | Motor | PID Control | Robot



สำนักพิมพ์ ส.ส.ท.

ไมโครคอนโทรลเลอร์

Arduino



โดย ผู้ช่วยศาสตราจารย์ดอนสัน ปงผาบ



สำนักพิมพ์ ส.ส.ท.

สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น)

250.-



ไมโครคอนโทรลเลอร์ Arduino

โดย... ผู้ช่วยศาสตราจารย์ดอนสัน ปงผาบ

ราคา 250 บาท

พิมพ์ครั้งที่ 1 มิถุนายน 2563

ข้อมูลทางบรรณานุกรมของสำนักหอสมุดแห่งชาติ

ดอนสัน ปงฝาบ.

ไมโครคอนโทรลเลอร์ Arduino. -- กรุงเทพฯ : สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2563.

240 หน้า.

1. ไมโครคอนโทรลเลอร์. I. ชื่อเรื่อง.

629.895

ISBN 978-974-443-779-2

จัดพิมพ์โดย **สำนักพิมพ์ ส.ส.ท.**
สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น)
5-7 ซอยสุขุมวิท 29 ถนนสุขุมวิท แขวงคลองเตยเหนือ เขตวัฒนา กรุงเทพฯ 10110
โทร. 0-2258-0320 (6 เลขหมายอัตโนมัติ), 0-2259-9160 (10 เลขหมายอัตโนมัติ)
ติดต่อสำนักพิมพ์ book4u@tpa.or.th
ติดต่อสั่งซื้อหนังสือ www.tpabook.com

จัดจำหน่ายโดย **บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)**
เลขที่ 1858/87-90 ถนนบางนา-ตราด แขวงบางนา เขตบางนา กรุงเทพฯ 10260
โทร. 0-2739-8000, 0-2739-8222 โทรสาร 0-2739-8356-9
www.se-ed.com

"ถ้าหนังสือมีข้อผิดพลาดเนื่องจากการพิมพ์ ให้นำมาแลกเปลี่ยนได้ที่สมาคมฯ" โทร. 0-2258-0320 ต่อ 1560, 1570

■**ผู้จัดการฝ่ายสำนักพิมพ์** สุกัญญา จารุการ **บรรณาธิการบริหาร** วัลภา ลีรัชดานนท์ **หัวหน้ากองบรรณาธิการ** ภาณิการ์ สุรรังสิกุล **บรรณาธิการต้นฉบับ** พรรณพิมล กิจไพฑูรย์ **กองบรรณาธิการ** อาทิตย์ นิ่มนวล, อภิชยา กู้ตลาด **ผู้จัดการแผนกศิลปกรรม** ศิรินุช อิศรางกูร ณ อยุธยา **ออกแบบปกและจัดรูปเล่ม** ธารินี คุตตะสิงคี **ศิลปกรรม** อนิล พยุงเกียรติคุณ **แผนกผลิตและการตลาดสำนักพิมพ์** รินดา คันธวร, แสงเงิน นาคพัฒน์

■ **พิมพ์ที่ :** หจก. ที.เอส.บี โปรดักส์

เกี่ยวกับ สำนักพิมพ์ ส.ส.ท.

สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น) หรือ ส.ส.ท. ก่อตั้งอย่างเป็นทางการในวันที่ 24 มกราคม พ.ศ. 2516 จากความตั้งใจมุ่งมั่น ความร่วมมือร่วมใจ และความเสียสละ ทุ่มเทกำลังกายและกำลังใจของกลุ่มบุคคลที่เคยไปศึกษาและดูงานโดยทุน ABK & AOTS ณ ประเทศญี่ปุ่น โดยมี ฯพณฯ สมหมาย ฮุนตระกูล เป็นประธานคณะกรรมการก่อตั้ง และสำเร็จด้วยความช่วยเหลืออย่างดีจาก อาจารย์ โงอิจิ โฮซูมิ อดีตประธานกรรมการ สมาคมความร่วมมือทางเศรษฐกิจญี่ปุ่น-ไทย (JTECS) ทั้งนี้โดยได้รับความช่วยเหลือทางด้านเงินทุนจากกระทรวงการค้าระหว่างประเทศและอุตสาหกรรม (METI) ประเทศญี่ปุ่น ซึ่งไม่มีพันธะผูกพันใด ๆ เพื่อใช้จ่ายในการดำเนินกิจกรรมต่าง ๆ

สำหรับ สำนักพิมพ์ ส.ส.ท. จัดตั้งขึ้นเมื่อ พ.ศ. 2516 พร้อม ๆ กับการก่อตั้งสมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น) โดยใช้ชื่อว่า โครงการตำรา เพื่อมุ่งส่งเสริมให้มีตำราภาษาไทยเกี่ยวกับเทคโนโลยีต่าง ๆ โดยเฉพาะอย่างยิ่งในระดับอาชีวศึกษา ซึ่งในขณะนั้นยังมีอยู่จำกัดให้แพร่หลายขึ้น เพื่อยกระดับมาตรฐานการศึกษาในสายวิชาชีพซึ่งเป็นกำลังสำคัญในการพัฒนาอุตสาหกรรมไทย

ในระยะ 4-5 ปี หลังจากการก่อตั้งสมาคมฯ โครงการตำราได้เปลี่ยนชื่อเป็น โครงการสนับสนุนเทคนิคอุตสาหกรรม และได้ขยายกิจกรรมเป็น ส่วนตำราสนับสนุนเทคนิคอุตสาหกรรม ในปี พ.ศ. 2539 พร้อม ๆ กับการขยายขอบข่ายหนังสือที่จัดพิมพ์ให้ครอบคลุมหนังสือด้านการบริหารจัดการธุรกิจอุตสาหกรรม และจิตวิทยา-การพัฒนาตนเอง รวมถึงคณิตศาสตร์ วิทยาศาสตร์ ทั้งที่เป็นงานเขียนและงานแปล ภายใต้ชื่อสำนักพิมพ์ ส.ส.ท. โดยมีวัตถุประสงค์เพื่อถ่ายทอดและเผยแพร่ความรู้ในสาขาต่าง ๆ ให้แก่บุคลากรทั้งภาครัฐและเอกชน ตลอดจนนักเรียน นักศึกษา เยาวชน ซึ่งจะเป็นรากฐานสำคัญในการพัฒนาอุตสาหกรรม เศรษฐกิจ และสังคมไทยให้เติบโตได้อย่างยั่งยืนต่อไป

สำนักพิมพ์ ส.ส.ท. ใคร่ขอแสดงความขอบคุณเป็นอย่างยิ่งต่อผู้เขียนและผู้แปลทุกท่านที่มีส่วนสำคัญในความสำเร็จของสำนักพิมพ์ตั้งแต่เริ่มต้นจวบจนปัจจุบัน และหวังว่าหนังสือของสำนักพิมพ์ ส.ส.ท. จะมีส่วนช่วยในการพัฒนาบุคลากร อันจะนำไปสู่การสร้างสรรค์สังคมและเศรษฐกิจของประเทศให้ก้าวหน้าและยั่งยืน และหากท่านผู้อ่านมีข้อชี้แนะประการใด ขอได้โปรดแจ้งให้ทางสำนักพิมพ์ทราบด้วย จักเป็นพระคุณยิ่ง

สำนักพิมพ์ ส.ส.ท.
สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น)

คำนำ

หนังสือ “ไมโครคอนโทรลเลอร์ Arduino” ใช้สำหรับประกอบการเรียนการสอนในวิชาที่เกี่ยวกับไมโครคอนโทรลเลอร์ Arduino ซึ่งเป็นไมโครคอนโทรลเลอร์ที่ได้รับความนิยมสูงสุดในปัจจุบัน เนื้อหาของหนังสือประกอบด้วย ความรู้พื้นฐานของไมโครคอนโทรลเลอร์ การเขียนโปรแกรม ดิจิทัลอินพุตเอาต์พุต แอนะล็อกอินพุตเอาต์พุต สเต็ปเปอร์มอเตอร์ ดีซีมอเตอร์ เซอร์โวมอเตอร์ เซนเซอร์ จอแสดงผล ระบบควบคุมแบบ PID หุ่นยนต์เดินตามเส้น หุ่นยนต์ 4 ขา และหุ่นยนต์ 2 ล้อสมดุล

ผู้เขียนเพียรพยายามอย่างเต็มกำลังความสามารถเพื่อให้หนังสือเล่มนี้สมบูรณ์ที่สุด โดยเรียบเรียงตามลำดับเนื้อหา อธิบายสรุปเป็นขั้นตอนให้เข้าใจง่าย ใช้รูปภาพประกอบการอธิบาย และเสริมตัวอย่างเพื่อเพิ่มความเข้าใจ จึงหวังว่าจะเป็นประโยชน์แก่ผู้เรียนจนสามารถนำความรู้ไปประยุกต์ใช้งานได้ต่อไป ทั้งนี้หากมีข้อบกพร่องหรือข้อผิดพลาดประการใด ผู้เขียนขอน้อมรับคำแนะนำติชมเพื่อการปรับปรุงแก้ไขและพัฒนาต่อไป

สุดท้าย ขอขอบพระคุณอาจารย์ทุกท่านที่ประสิทธิ์ประสาทความรู้ และสมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น) ที่สนับสนุนการจัดพิมพ์จนหนังสือเล่มนี้สำเร็จเป็นรูปเล่ม

ผู้ช่วยศาสตราจารย์ดอนสัน ปงผาบ

สารบัญ

บทที่

1 พื้นฐานไมโครคอนโทรลเลอร์



ในปัจจุบันเทคโนโลยีทางด้านอิเล็กทรอนิกส์และไมโครคอนโทรลเลอร์ได้พัฒนาอย่างรวดเร็วแบบก้าวกระโดดตามจำนวนทรานซิสเตอร์ในซีพียู (CPU : Central Processing Unit) ที่เพิ่มขึ้นประมาณเท่าตัวทุก ๆ 2 ปี ตามกฎของมัวร์ (Moore's Law) ที่กล่าวไว้เมื่อปี ค.ศ. 1965 กฎนี้ได้ถูกพิสูจน์มาแล้วมากกว่าครึ่งศตวรรษและเห็นผลอย่างชัดเจนใน 10 ปีที่ผ่านมา ทำให้เทคโนโลยีและอุปกรณ์อิเล็กทรอนิกส์เกิดการเปลี่ยนแปลงอย่างรวดเร็ว เช่น กล้องดิจิทัลที่มีความคมชัดและมีฟังก์ชันปรับแต่งรูปภาพได้หลากหลาย สมาร์ทโฟน คอมพิวเตอร์ที่มีขนาดเล็กแต่มีความสามารถมากขึ้น เครื่องใช้ไฟฟ้าที่ทำงานเชื่อมต่อกันได้ รถยนต์ไร้คนขับ ซึ่งการเปลี่ยนแปลงเหล่านี้ก้าวหน้าตามความสามารถของซีพียูและไมโครคอนโทรลเลอร์

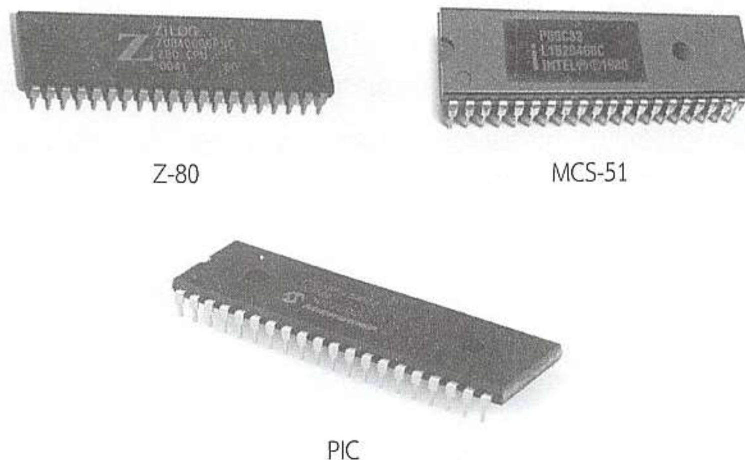
1.1 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ (Microcontroller) คือตัวควบคุมขนาดเล็ก เป็นอุปกรณ์อิเล็กทรอนิกส์ชนิดหนึ่งที่ใช้เทคโนโลยีการผลิตระดับไมโคร ถูกออกแบบมาเพื่อใช้สำหรับประมวลผลและควบคุมการทำงานของเครื่องอัตโนมัติต่าง ๆ เช่น เครื่องหยอดเหรียญ เครื่องขายของอัตโนมัติ หุ่นยนต์ แขนกล และเครื่องใช้ไฟฟ้า ไมโครคอนโทรลเลอร์มีหลากหลายตระกูล แต่ที่ได้รับความนิยมในอดีตที่ผ่านมามีดังนี้

1) Z-80 เป็นไมโครโพรเซสเซอร์ขนาด 8 บิต ผลิตโดยบริษัทไซล็อก (Zilog) ไม่มีพอร์ตอินพุตและเอาต์พุตในตัว มีบัสข้อมูล (Data Bus) ขนาด 8 บิต และบัสตำแหน่งที่อยู่ (Address Bus) ขนาด 16 บิต สามารถเชื่อมต่อกับหน่วยความจำขนาด 64 กิโลไบต์ เขียนโปรแกรมควบคุมด้วยภาษาแอสเซมบลีหรือป้อนรหัสภาษาเครื่อง Z-80 เริ่มเป็นที่นิยมราวคริสต์ทศวรรษ 1980 เป็นต้นมา และใช้ในการเรียนการสอนในอดีตที่ผ่านมา

2) MCS-51 เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิตของบริษัทอินเทล (Intel) มีพอร์ตอินพุตและเอาต์พุตในตัว สามารถเข้าถึงข้อมูลระดับบิตได้ เขียนโปรแกรมควบคุมด้วยภาษาแอสเซมบลีและภาษาซี เริ่มเป็นที่นิยมราวคริสต์ทศวรรษ 1990 โดยมีบริษัทผู้ผลิตหลายราย เช่น อินเทล แอทเมล (Atmel) ฟิลิปส์ (Philips) และซีเมนส์ (Siemens)

3) PIC (Peripheral Interface Controller) เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิตของบริษัทไมโครชิพ เทคโนโลยี (Microchip Technology) มีหน่วยความจำแบบแฟลช (Flash Memory) เป็นไมโครคอนโทรลเลอร์ที่รวมอุปกรณ์ต่าง ๆ เช่น หน่วยความจำรอม หน่วยความจำแรม A/D และบัสแบบ I2C ไว้ด้วยกันโดยไม่ต้องต่อกับอุปกรณ์ภายนอกทำให้สะดวกในการใช้งาน PIC เริ่มเป็นที่นิยมราวคริสต์ทศวรรษ 2000



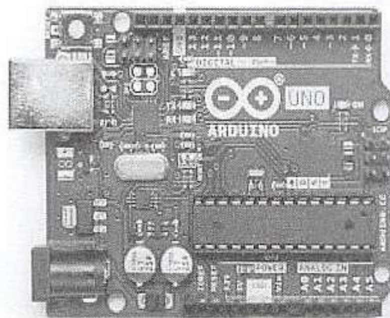
Z-80 MCS-51

PIC

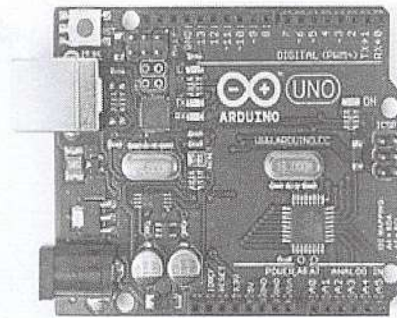
รูปที่ 1.1 ไมโครโพรเซสเซอร์และไมโครคอนโทรลเลอร์

1.2 Arduino คืออะไร

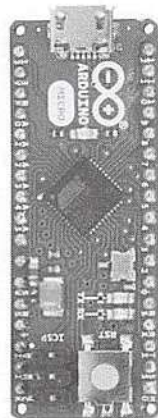
Arduino (อาดูอิโน่เป็นภาษาอิตาเลียน) คือไมโครคอนโทรลเลอร์ขนาด 8 บิต อยู่ในตระกูล AVR ผลิตโดยบริษัท Atmel ไมโครคอนโทรลเลอร์ตระกูล AVR มีสถาปัตยกรรมแบบ RISC (Reduced Instruction Set Computer) มีความเร็วในการประมวลผล 1 คำสั่งต่อ 1 สัญญาณนาฬิกา ถูกออกแบบมาเพื่อให้ใช้งานง่าย สามารถเชื่อมต่อกับคอมพิวเตอร์ด้วยพอร์ต USB มีบอร์ดอุปกรณ์เชื่อมต่อหรือ Arduino Shield มากมาย ทำให้สะดวกในการพัฒนางาน การออกแบบและพัฒนาได้เปิดเผยข้อมูลทั้งด้านฮาร์ดแวร์และซอฟต์แวร์ (Open Source) จึงมีผู้ใช้งานจำนวนมาก Arduino ถูกออกแบบมาให้ใช้งานง่ายทั้งด้านฮาร์ดแวร์และการเขียนโปรแกรม จึงเหมาะสำหรับทุกคนที่สนใจและต้องการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ บอร์ด Arduino มีหลากหลายรุ่น เช่น Arduino UNO R3/SMD, Arduino Micro และ Arduino MEGA แสดงดังรูปที่ 1.2



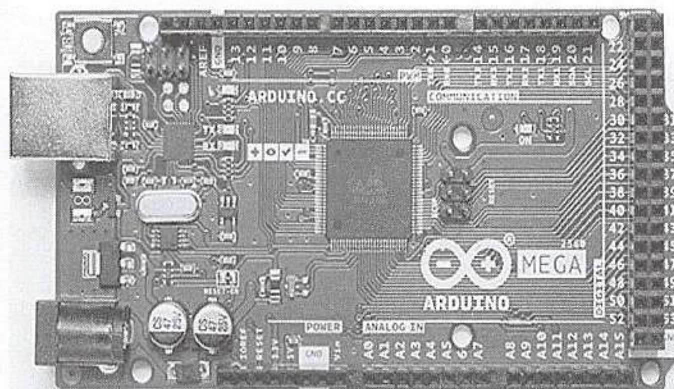
Arduino UNO R3



Arduino UNO SMD



Arduino Micro



Arduino MEGA 2560

รูปที่ 1.2 รุ่นต่าง ๆ ของบอร์ด Arduino

Arduino Uno R3 คือบอร์ดไมโครคอนโทรลเลอร์ ATmega328P ขนาด 8 บิต เป็นบอร์ดที่ได้รับความนิยมอย่างมากเนื่องจากใช้งานง่าย มีขาสัญญาณแอนะล็อก 6 อินพุต ขาสัญญาณดิจิทัล 13 ขา ไลบรารีและอุปกรณ์เชื่อมต่อส่วนใหญ่สามารถใช้ร่วมกับ Arduino Uno ได้



รูปที่ 1.3 บอร์ด Arduio UNO R3

คุณลักษณะของบอร์ด

ไมโครคอนโทรลเลอร์	ATmega328P
แรงดันไฟฟ้า	5 V
แรงดันไฟฟ้าอินพุต (แนะนำ)	7–12 V
แรงดันไฟฟ้าอินพุต (จำกัด)	6–20 V
ดิจิทัลอินพุตและเอาต์พุต	14 บิต (6 เอาต์พุต PWM)
แอนะล็อกอินพุต	6 บิต
แรงดันและกระแสไฟที่จ่ายได้ในแต่ละบิต	5 V 40 mA
แรงดันและกระแสไฟที่จ่ายได้ในแต่ละบิต	3.3 V 50 mA
หน่วยความจำแบบแฟลช (Flash)	32 KB (500B Boot Loader)
หน่วยความจำแรม (RAM)	2 KB
หน่วยความจำรอม (ROM)	1 KB
ความถี่	16 MHz
ขนาด	68.6 x 53.4 mm
น้ำหนัก	25 กรัม

ขาสัญญาณของบอร์ด Arduino UNO และหน้าที่

3V3 คือ ขาสัญญาณแรงดันไฟ 3.3 V

5V คือ ขาสัญญาณแรงดันไฟ 5 V

GND คือ ขากราวด์

Vin คือ ขาแรงดันไฟอินพุตที่ป้อนให้บอร์ด (Power Jack)

A0–A5 คือ ขาสัญญาณแอนะล็อกอินพุต มี 6 ขา คือ A0 A1 A2 A3 A4 และ A5 มีความละเอียด 10 บิต มีค่าอยู่ในช่วง 0-1023

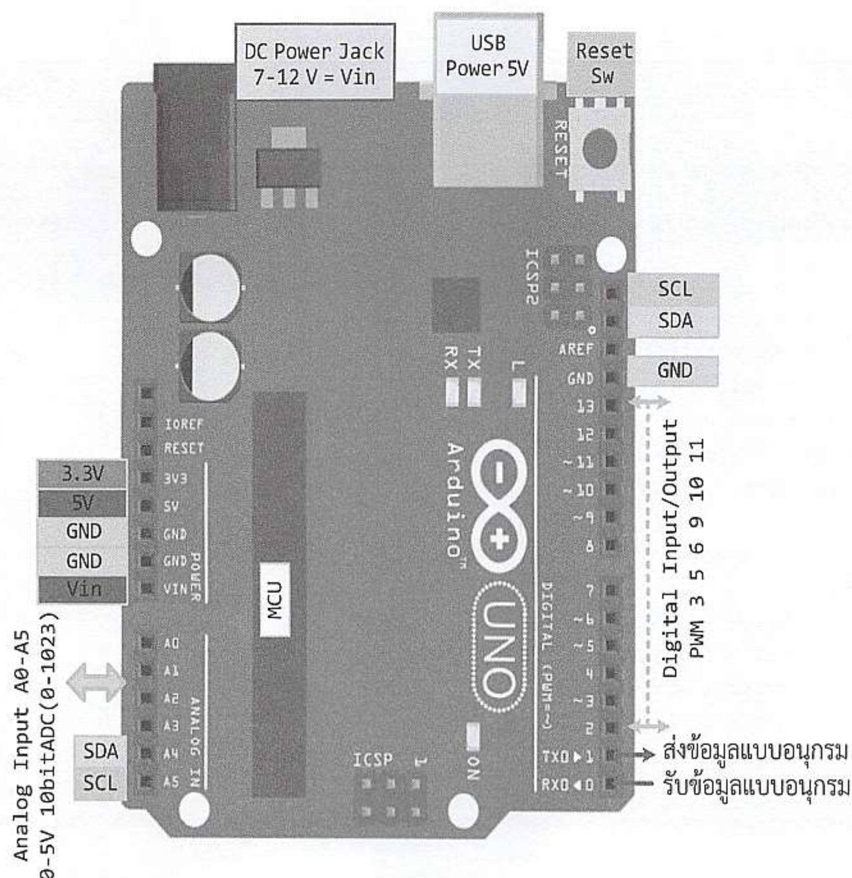
SDA(A4) คือ ขาสัญญาณข้อมูล

SCL(A5) คือ ขาสัญญาณนาฬิกา ใช้ในการติดต่อสื่อการแบบ I2C เช่น การเชื่อมต่อกับจอแสดงผล LCD ในบอร์ดจะมีอยู่ 2 จุด

RXD คือ ขาสัญญาณรับข้อมูลของการสื่อสารพอร์ตอนุกรม

TXD คือ ขาสัญญาณส่งข้อมูลของการสื่อสารพอร์ตอนุกรม

2–13 คือ ขาสัญญาณดิจิทัลอินพุต/เอาต์พุต และมีเอาต์พุต PWM 6 ขา คือ 3 5 6 9 10 และ 11



รูปที่ 1.4 ขาสัญญาณของบอร์ด Arduino UNO

1.3 โปรแกรม Arduino

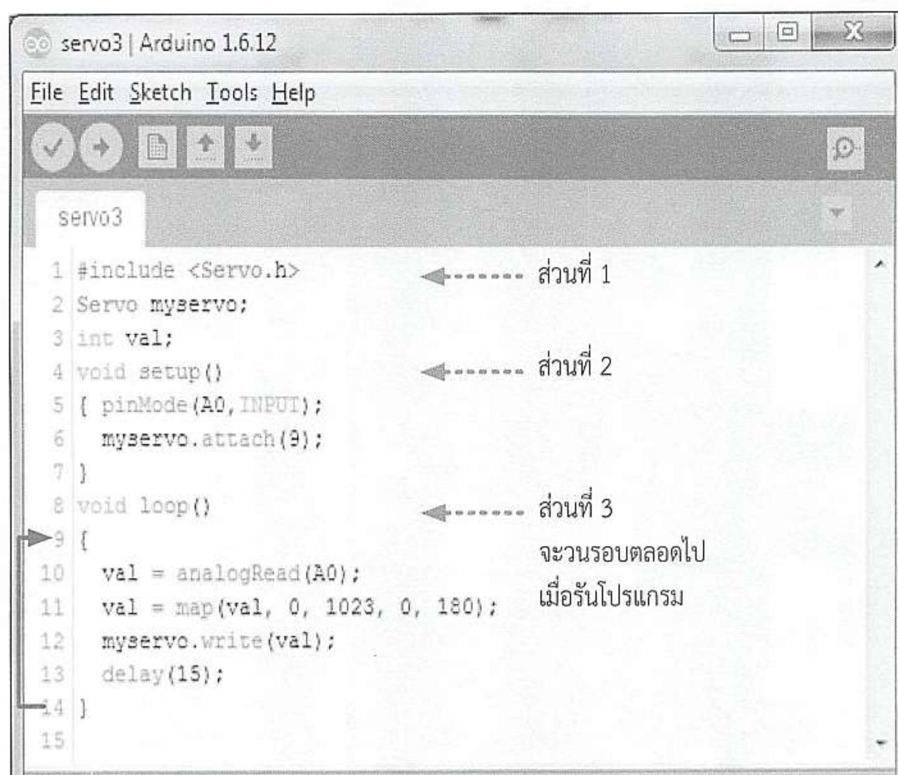
Arduino IDE (Integrated Development Environment) คือซอฟต์แวร์ที่ใช้เขียนหรือพัฒนาโปรแกรม แบ่งได้เป็น 3 ส่วนคือ

1) ไฟล์หัวโปรแกรม ซึ่งจะเขียนก็ต่อเมื่อมีการเรียกใช้ไลบรารี
2) void setup() เป็นส่วนของการกำหนดค่า
3) void loop() เป็นส่วนของตัวโปรแกรม

ส่วนที่ 1 ไฟล์หัวโปรแกรม จะประกาศก็ต่อเมื่อมีการเรียกใช้งานไลบรารีของ Arduino ตัวอย่างเช่น #include <Servo.h> หมายถึง เรียกใช้งานไลบรารี Servo.h เพื่อใช้งานเกี่ยวกับเซอร์โวมอเตอร์

ส่วนที่ 2 void setup() คือการกำหนดค่าเริ่มต้นของโปรแกรม เช่น การกำหนดขาสัญญาณ การกำหนดอัตราการสื่อสารแบบอนุกรม หรือการกำหนดขาอินพุต/เอาต์พุต ซึ่งจะทำงานเพียงครั้งเดียว

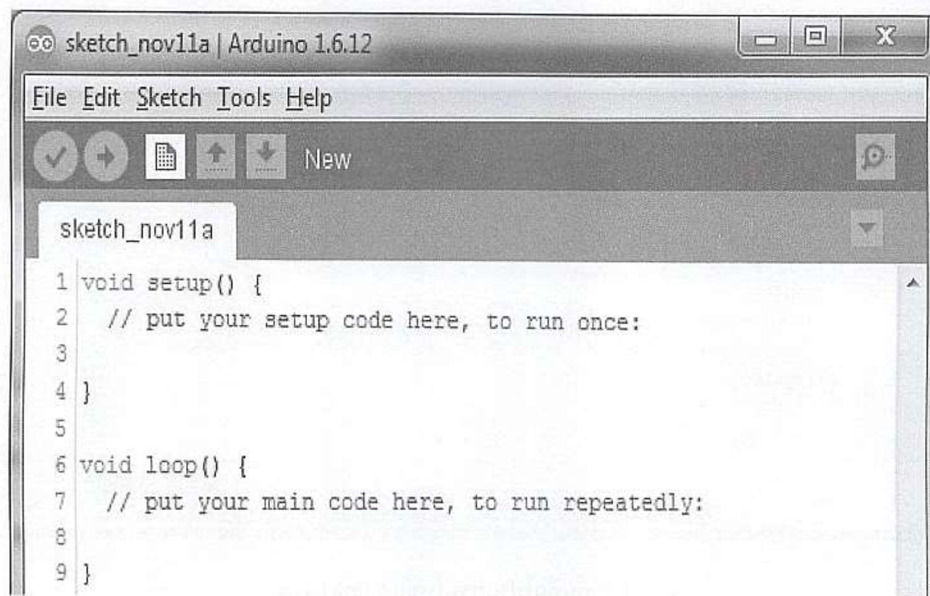
ส่วนที่ 3 void loop() คือโปรแกรมหลักที่จะทำงานตลอดเวลาหรือวนลูปทำงานไปตลอด ส่วนนี้อยู่ระหว่างเครื่องหมายปีกกาเปิดถึงเครื่องหมายปีกกาปิด และโปรแกรมจะหยุดทำงานก็ต่อเมื่อปิดสวิตช์ ภายในฟังก์ชัน void loop() ประกอบไปด้วยตัวแปร ชุดคำสั่ง และฟังก์ชันต่าง ๆ โดยการเขียนโปรแกรมตามโครงสร้างของภาษาซี



รูปที่ 1.5 ส่วนประกอบของโปรแกรม Arduino

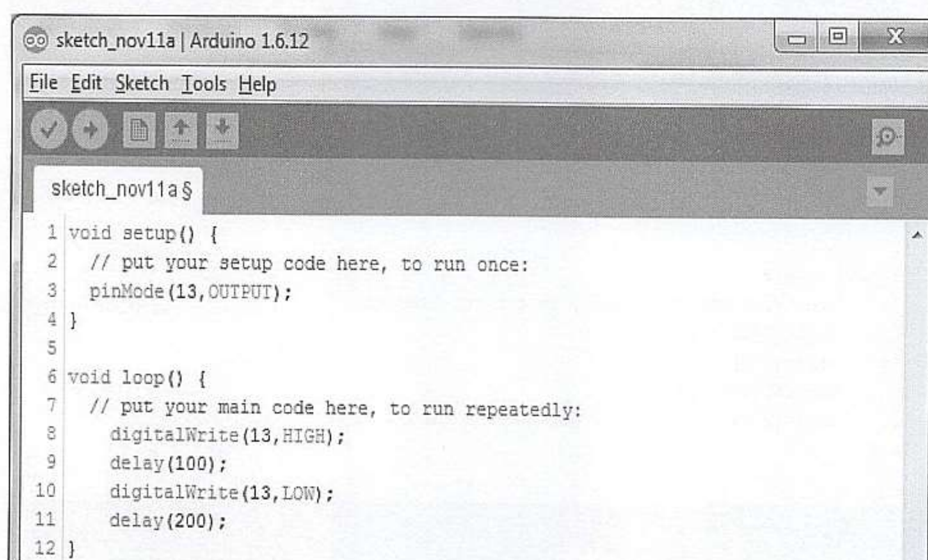
การใช้งานโปรแกรม Arduino IDE เริ่มจากการสร้างไฟล์ใหม่ เขียนโปรแกรมตามโครงสร้างของภาษาซี คอมไพล์และโหลดโปรแกรมลงบนบอร์ด Arduino ซึ่งมีขั้นตอนดังนี้

ขั้นตอนที่ 1 เริ่มจากเข้าโปรแกรม Arduino IDE แล้วกด New จะได้หน้าต่างไฟล์ใหม่ดังนี้



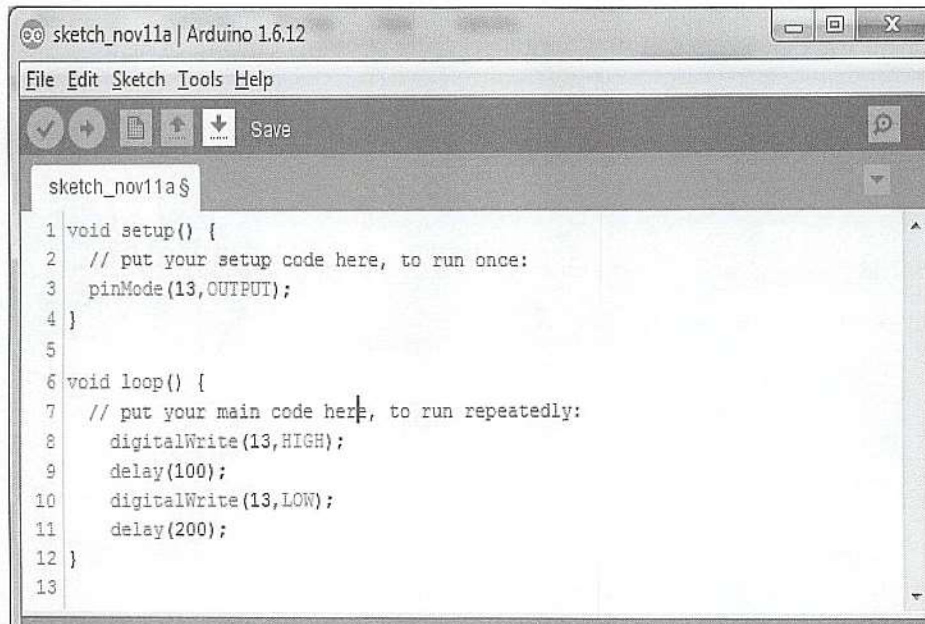
รูปที่ 1.6 หน้าต่างโปรแกรม Arduino

ขั้นตอนที่ 2 เขียนโปรแกรมไฟกะพริบตามตัวอย่าง

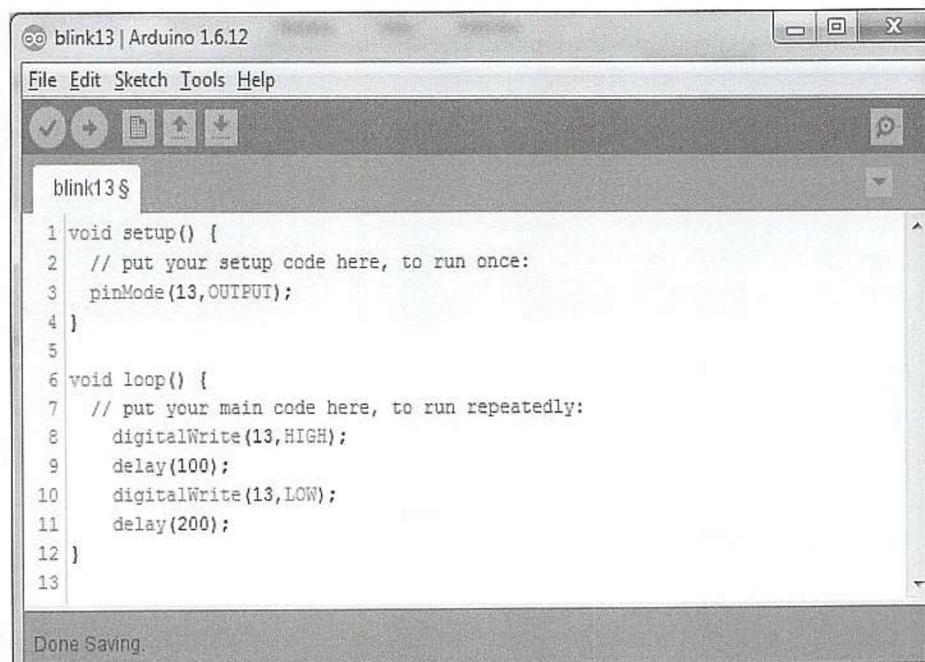


รูปที่ 1.7 ตัวอย่างโปรแกรมไฟกะพริบ

ขั้นตอนที่ 3 บันทึกโปรแกรมในชื่อ blink13.ino เมื่อบันทึกเสร็จแล้วโปรแกรมจะเปลี่ยนชื่อเป็น blink13 สังเกตชื่อโปรแกรมบริเวณด้านบนซ้ายดังรูปที่ 1.9

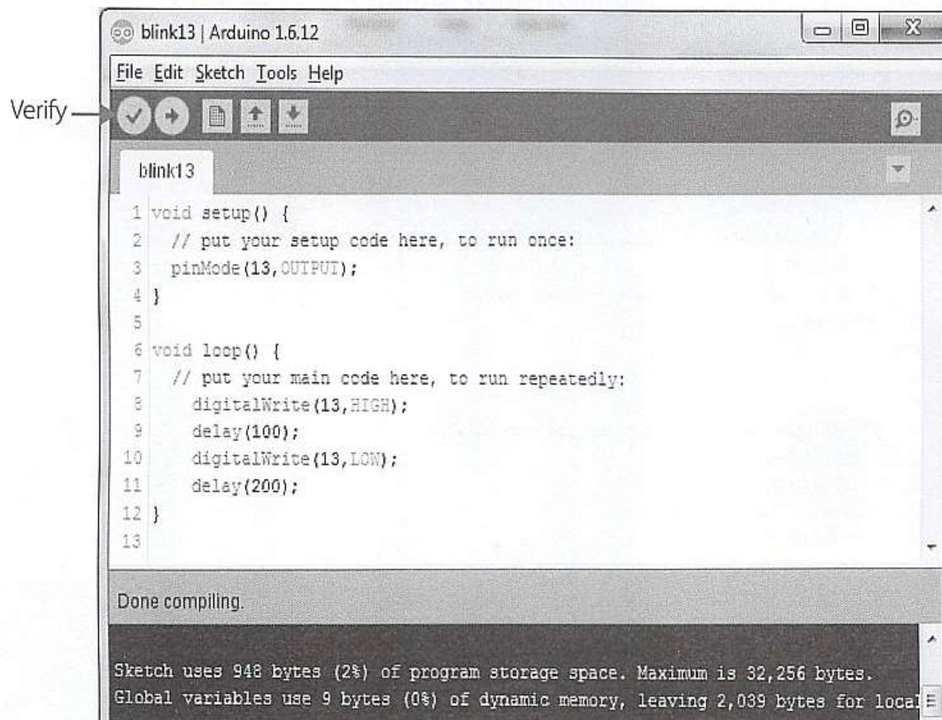


รูปที่ 1.8 บันทึกโปรแกรมในชื่อ blink13.ino

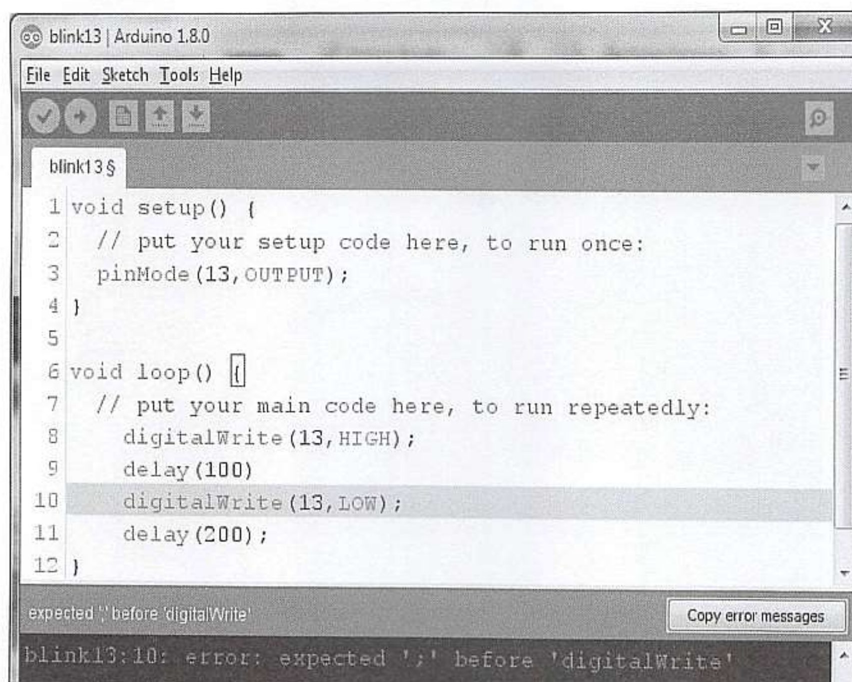


รูปที่ 1.9 หน้าต่างโปรแกรม blink13.ino

ขั้นตอนที่ 4 ทำการ Verify หรือคอมไพล์เพื่อตรวจสอบความถูกต้อง ขั้นตอนนี้ไม่จำเป็นต้องมีบอร์ด Arduino หากเกิดข้อผิดพลาดต้องกลับไปแก้ไขที่โปรแกรม

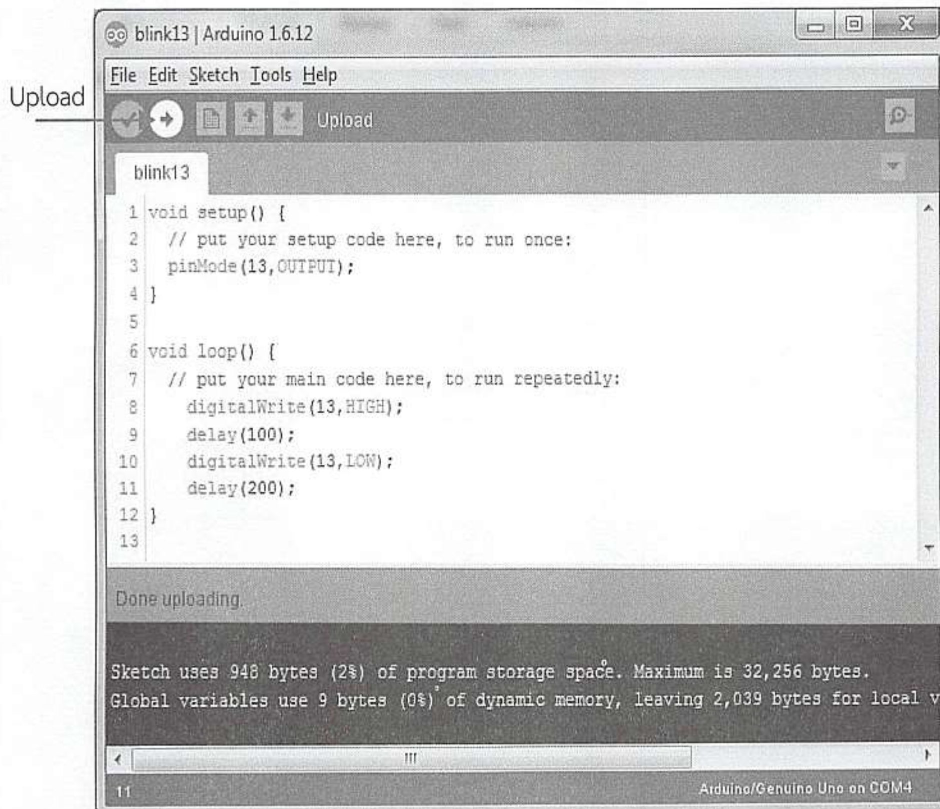


รูปที่ 1.10 ผลการคอมไพล์โปรแกรมผ่าน

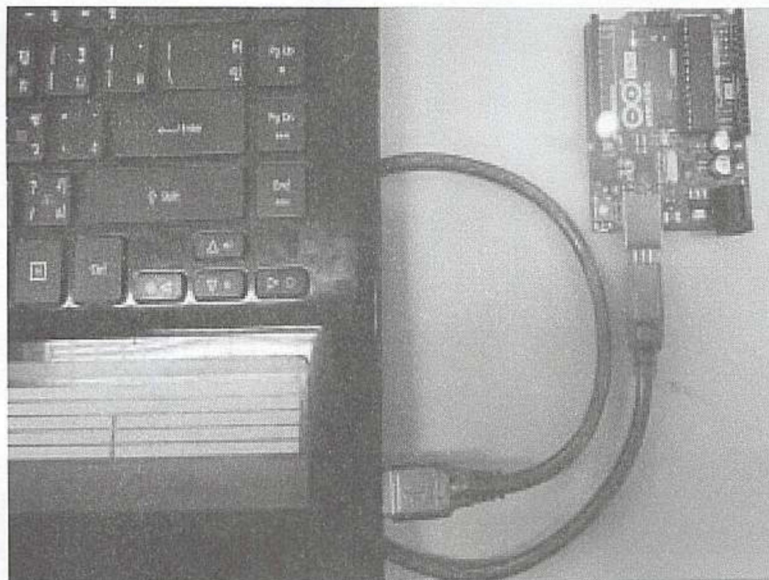


รูปที่ 1.11 ผลการคอมไพล์โปรแกรมเกิดข้อผิดพลาด

ขั้นตอนที่ 5 เชื่อมต่อบอร์ด Arduino กับคอมพิวเตอร์แล้วโหลดโปรแกรมลงบอร์ด (Upload) ให้สังเกตไฟที่บอร์ดจะกะพริบแสดงว่ากำลังโหลดข้อมูลลงบอร์ด เมื่อโหลดโปรแกรมเสร็จแล้ว ขา 13 ของบอร์ดจะมีไฟกะพริบ หากเกิดข้อผิดพลาดต้องกลับไปแก้ไขที่โปรแกรมหรือแก้ไขที่พอร์ตอนุกรม



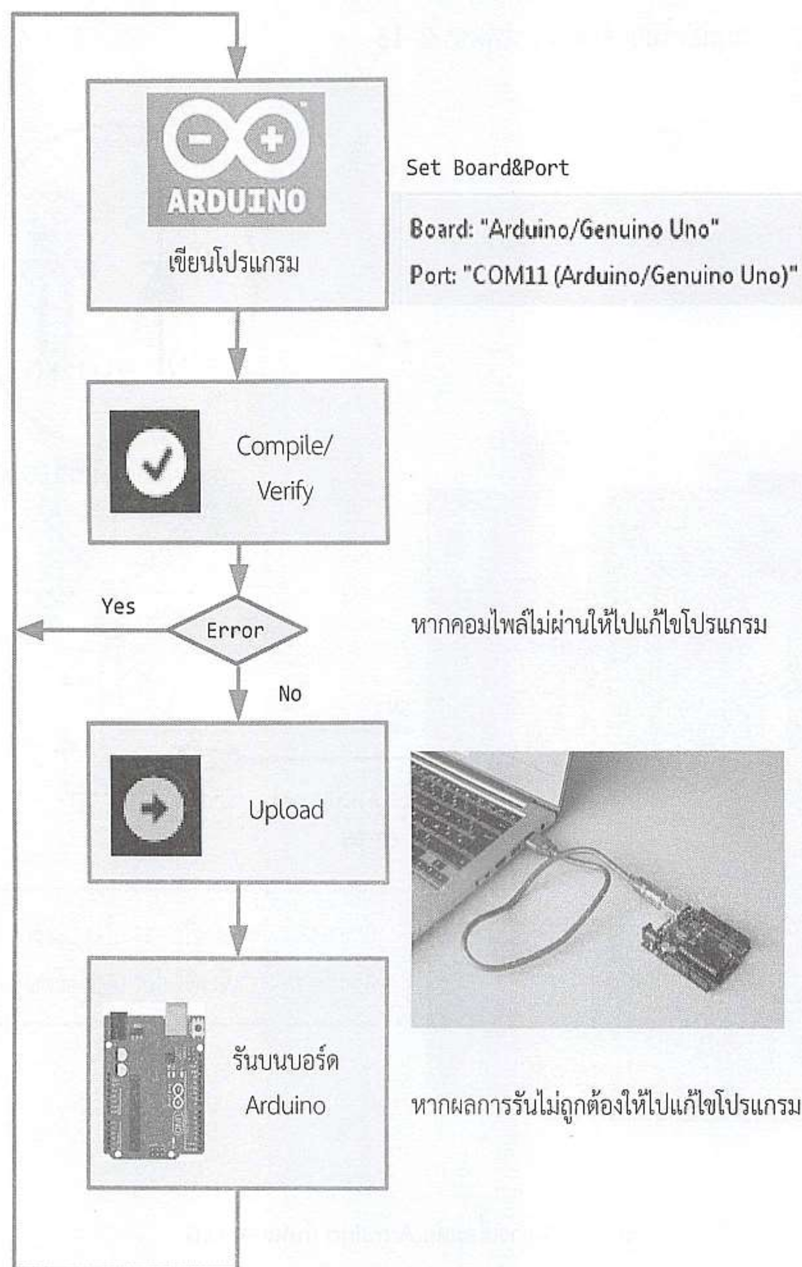
รูปที่ 1.12 โหลดโปรแกรมลงบอร์ด



รูปที่ 1.13 ไฟกะพริบที่ขา 13 ของบอร์ด Arduino

1.4 ขั้นตอนการพัฒนาโปรแกรม

ขั้นตอนการพัฒนาโปรแกรมเริ่มจากการใช้โปรแกรม Arduino IDE เพื่อเขียนโปรแกรม เมื่อเสร็จแล้วให้กำหนดรุ่นของบอร์ด Arduino และหมายเลขพอร์ตของการเชื่อมต่อ จากนั้นทำการคอมไพล์หรือ Verify หากเกิดข้อผิดพลาดต้องกลับไปแก้ไขโปรแกรมใหม่ ถ้าไม่เกิดข้อผิดพลาดก็สามารถโหลดโปรแกรมลงบอร์ด Arduino และรันโปรแกรมได้

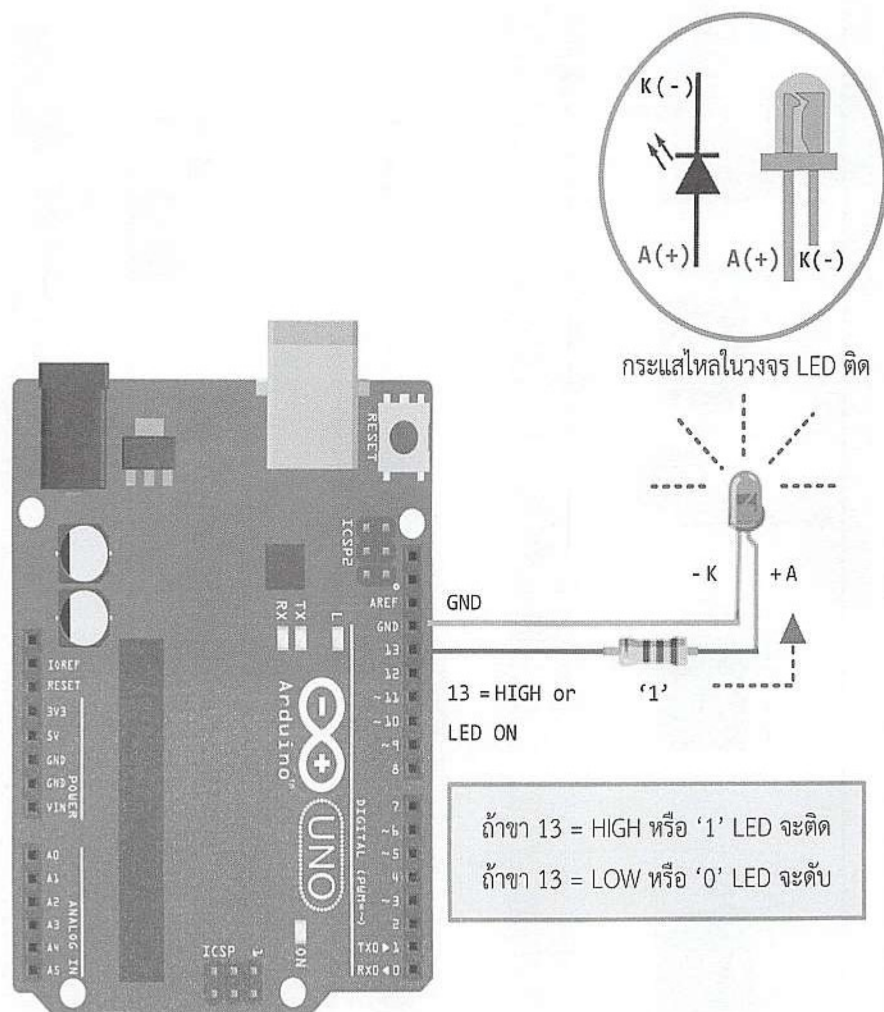


รูปที่ 1.14 ขั้นตอนการพัฒนาโปรแกรม

1.5 หลอดแสดงผล LED

หลอดแสดงผล LED (Light Emitting Diode) หรือไดโอดเปล่งแสง เป็นอุปกรณ์อิเล็กทรอนิกส์ที่ใช้ในการแสดงผลไฟติดหรือดับ LED มี 2 ขา คือ ขาแอโนด (Anode) ต้องป้อนไฟบวก และขาแคโทด (K) เป็นขากราวด์หรือไฟลบ

การทำงานของวงจร เมื่อสัญญาณเอาต์พุตเป็นลอจิก 1 หรือสัญญาณไฟ 5 โวลต์ จะมีกระแสไหลในวงจรทำให้ไฟติด ส่วนตัวต้านทานจะทำหน้าที่ลดกระแสไฟฟ้าที่ไหลในวงจรให้เหมาะสม การเชื่อมต่อหลอดแสดงผล LED สามารถต่อได้ที่ขาดิจิทัลเอาต์พุตขา 2–13

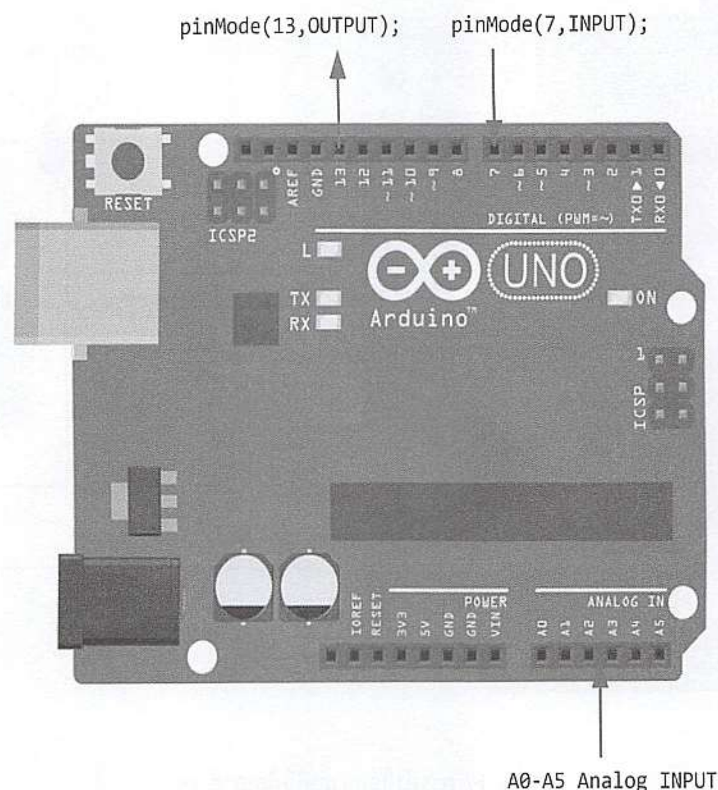


รูปที่ 1.15 การเชื่อมต่อ Arduino กับหลอด LED

1.6 ฟังก์ชัน pinMode();

ในการใช้งานขาสัญญาณของไมโครคอนโทรลเลอร์จำเป็นต้องกำหนดสถานะของขาสัญญาณให้เป็นอินพุตหรือเอาต์พุตก่อน เนื่องจากขาสัญญาณดิจิทัลของไมโครคอนโทรลเลอร์สามารถเป็นได้ทั้งอินพุตและเอาต์พุต โดยจะกำหนดเป็นอินพุตก็ต่อเมื่อต่อกับอุปกรณ์อินพุต เช่น สวิตช์ เซนเซอร์ และกำหนดเป็นเอาต์พุตเมื่อต่อกับอุปกรณ์เอาต์พุต เช่น LED รีเลย์ การกำหนดสถานะดังกล่าวจะใช้ฟังก์ชัน pinMode();

pinMode(); คือฟังก์ชันกำหนดขาสัญญาณให้มีสถานะเป็นอินพุตหรือเอาต์พุต



รูปที่ 1.16 การกำหนดขาอินพุตและเอาต์พุต

ตัวอย่าง

pinMode(13,OUTPUT);

หมายถึง กำหนดให้ขา 13 เป็นขาเอาต์พุตเพื่อส่งสัญญาณออกไปควบคุมอุปกรณ์

pinMode(7,INPUT);

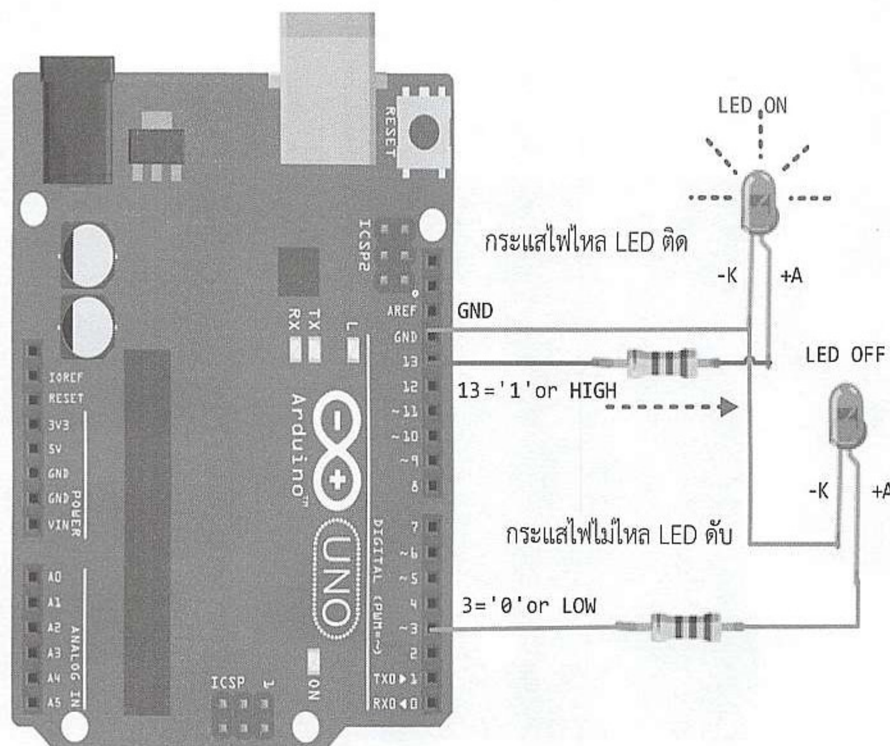
หมายถึง กำหนดให้ขา 7 เป็นขาอินพุตเพื่อรับข้อมูลจากสวิตช์หรือเซนเซอร์

นอกจากนี้ ขาสัญญาณ A0–A5 เป็นพอร์ตแอนะล็อกอินพุตอยู่แล้ว จึงไม่จำเป็นต้องใช้ฟังก์ชัน pinMode(); เพื่อกำหนดให้เป็นอินพุต

1.7 ฟังก์ชัน digitalWrite();

ในการควบคุมอุปกรณ์ดิจิทัลเอาต์พุตต้องส่งสัญญาณลอจิก 0 หรือ LOW และสัญญาณลอจิก 1 หรือ HIGH ซึ่งเป็นระบบตัวเลขฐานสอง โดยสัญญาณไฟ 0 โวลต์แทนสัญญาณลอจิก 0 และสัญญาณไฟ 5 โวลต์แทนสัญญาณลอจิก 1 ในการส่งสัญญาณดิจิทัลเอาต์พุตจะใช้ฟังก์ชัน digitalWrite();

digitalWrite(); คือฟังก์ชันสำหรับส่งข้อมูลดิจิทัล 0 (LOW) หรือ 1 (HIGH) ออกทางขาเอาต์พุต



รูปที่ 1.17 การส่งสัญญาณดิจิทัลเอาต์พุต

ตัวอย่าง

```
digitalWrite(13,HIGH);
```

หมายถึง ให้เอาต์พุตขา 13 เป็นลอจิก 1 หรือส่งสัญญาณไฟ 5 โวลต์ทำให้ LED ติด

```
digitalWrite(13,1);
```

หมายถึง ให้เอาต์พุตขา 13 เป็นลอจิก 1 หรือส่งสัญญาณไฟ 5 โวลต์ สามารถใช้ 1 แทน HIGH ได้

```
digitalWrite(3,0);
```

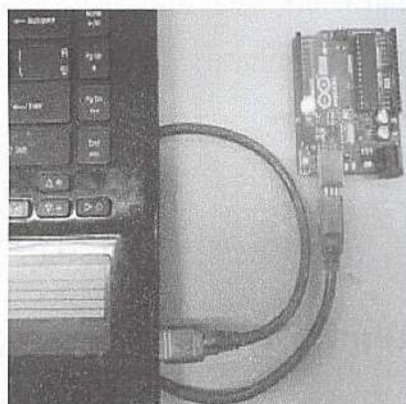
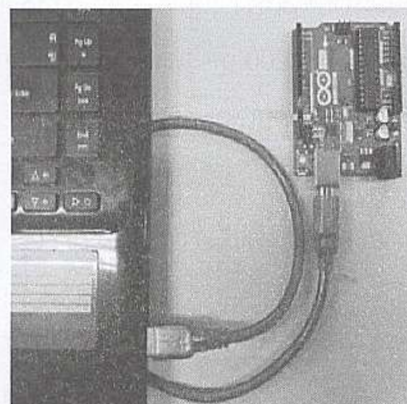
หมายถึง ให้เอาต์พุตขา 3 เป็นลอจิก 0 (LOW) หรือส่งสัญญาณไฟ 0 โวลต์ทำให้ LED ดับ

ตัวอย่างที่ 1.1 โปรแกรมไฟกะพริบที่ขา 13

```
void setup()                    //การกำหนดค่า
{
pinMode(13,OUTPUT);             //กำหนดให้ขา 13 เป็นเอาต์พุต
}
void loop()                     //วนลูปไปตลอด
{                               //เริ่ม
digitalWrite(13,HIGH);          //ขา 13 เป็น HIGH หลอด LED ติด
delay(200);                     //หน่วงเวลา 200 ms
digitalWrite(13,LOW);           //ขา 13 เป็น LOW หลอด LED ดับ
delay(500);                     //หน่วงเวลา 500 ms
}                               //จบ
```

ผลการรันโปรแกรม

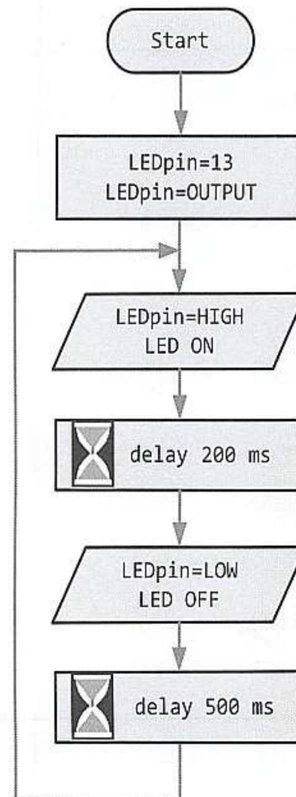
โปรแกรมจะส่งสัญญาณ HIGH หรือลอจิก 1 เป็นเวลา 200 ms และสัญญาณ LOW หรือลอจิก 0 เป็นเวลา 500 ms ออกที่ขา 13 ทำให้ LED ติดดับสลับกัน เกิดเป็นไฟกะพริบที่บอร์ด Arduino

รูปที่ 1.18 ไฟกะพริบที่ขา 13

ตัวอย่างที่ 1.2 โปรแกรมไฟกะพริบแบบประกาศตัวแปร

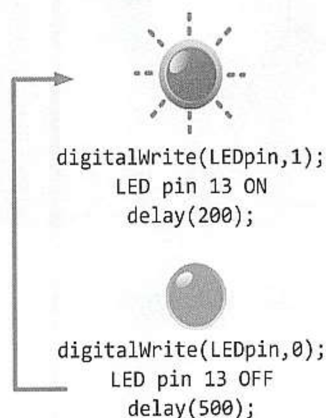
```
int LEDpin=13;
void setup()
{
pinMode(LEDpin,OUTPUT);
}
void loop()
{
digitalWrite(LEDpin,1);
delay(200);
digitalWrite(LEDpin,0);
delay(500);
}
```



รูปที่ 1.19 โฟลว์ชาร์ตไฟกะพริบที่ขา 13

ผลการรันโปรแกรม

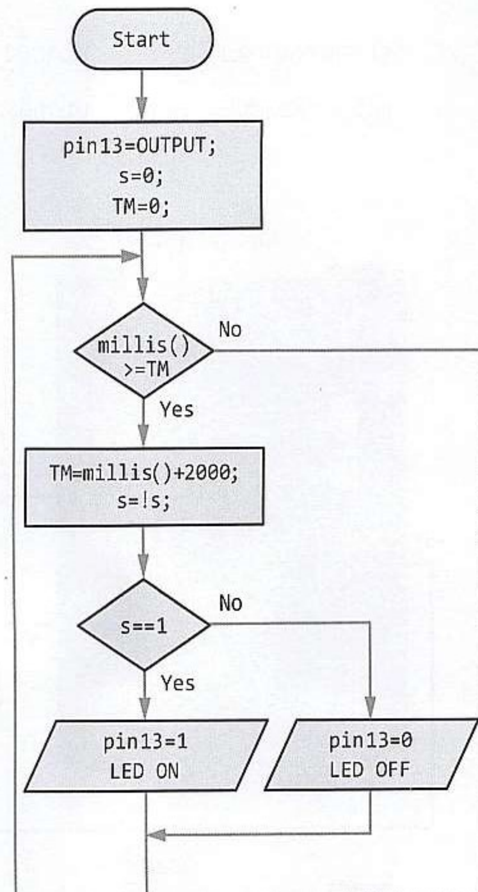
เมื่อประกาศตัวแปร LEDpin เท่ากับขา 13 ของไมโครคอนโทรลเลอร์ โปรแกรมจะส่งสัญญาณลอจิก 1 หรือ HIGH เป็นเวลา 200 ms และสัญญาณลอจิก 0 หรือ LOW เป็นเวลา 500 ms ออกที่ขา 13 ทำให้ LED ติดดับสลับกันเกิดเป็นไฟกะพริบ



รูปที่ 1.20 การเกิดไฟกะพริบที่ขา 13

ตัวอย่างที่ 1.3 โปรแกรมควบคุมไฟกะพริบโดยใช้ฟังก์ชัน millis();

```
unsigned long TM;
int s;
void setup()
{
  pinMode(13,OUTPUT);
}
void loop()
{
  if(millis()>=TM)
  {
    TM=millis()+2000;
    s=!s;
    if(s==1)
      digitalWrite(13,1);
    else
      digitalWrite(13,0);
  }
}
```



รูปที่ 1.21 โฟลว์ชาร์ตไฟกะพริบโดยใช้ฟังก์ชัน millis();

ผลการรันโปรแกรม

หลอด LED ที่ขา 13 จะกะพริบหรือติดดับทุก ๆ 2 วินาที (หรือ 2000 มิลลิวินาที) โปรแกรมจะจับเวลาการทำงานโดยเปรียบเทียบค่าฟังก์ชัน millis(); กับตัวแปร TM

หมายเหตุ

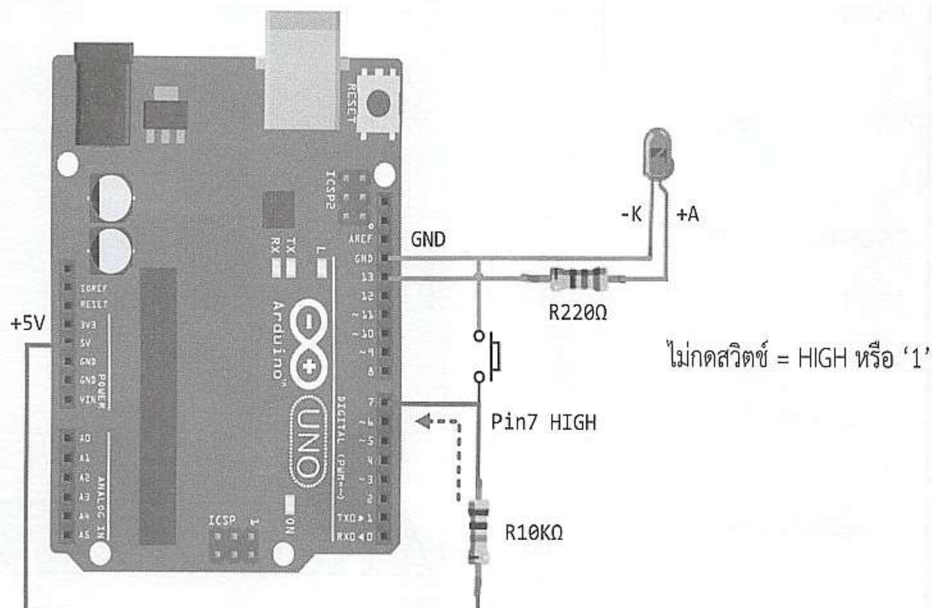
millis(); เป็นฟังก์ชันเกี่ยวกับเวลาในการรันโปรแกรม มีหน่วยเป็นมิลลิวินาที โดยจะเก็บค่าได้สูงสุดเป็น unsigned long ขนาด 32 บิต ฟังก์ชัน millis(); จะเพิ่มขึ้นทีละหนึ่งทุก ๆ มิลลิวินาทีโดยอัตโนมัติเมื่อรันโปรแกรม

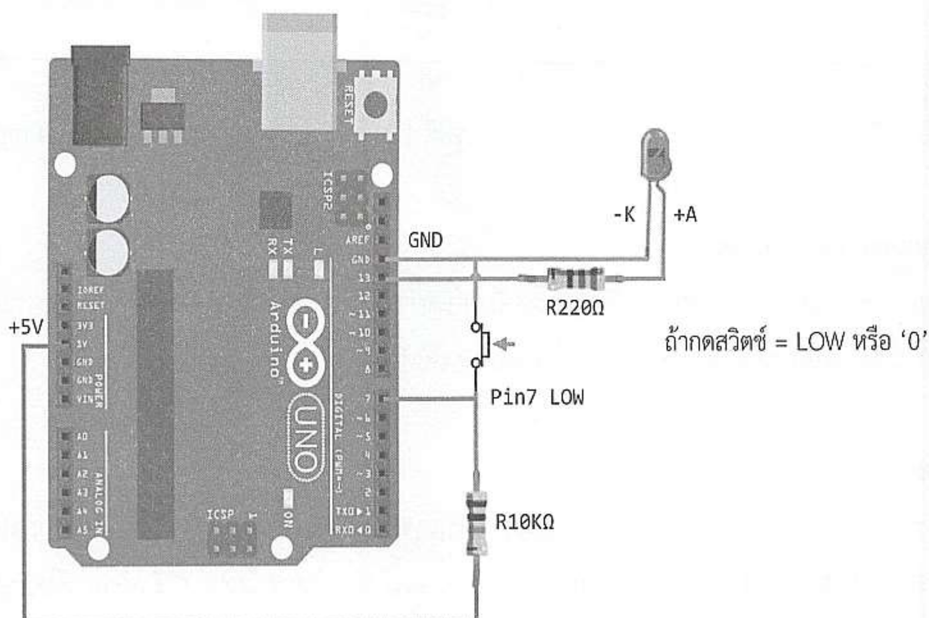
1.8 ฟังก์ชัน digitalRead();

การอ่านค่าสัญญาณดิจิทัลจากขาสัญญาณ Arduino สามารถอ่านได้จากขา 0-13 โดยใช้ฟังก์ชัน digitalRead(); ซึ่งค่าที่ได้จะมีค่าเป็นลอจิก 0 หรือ 1 เท่านั้น

digitalRead(); คือฟังก์ชันอ่านค่าข้อมูลดิจิทัลซึ่งสามารถอ่านได้ทั้งขาอินพุตและเอาต์พุต ตัวอย่างเช่น

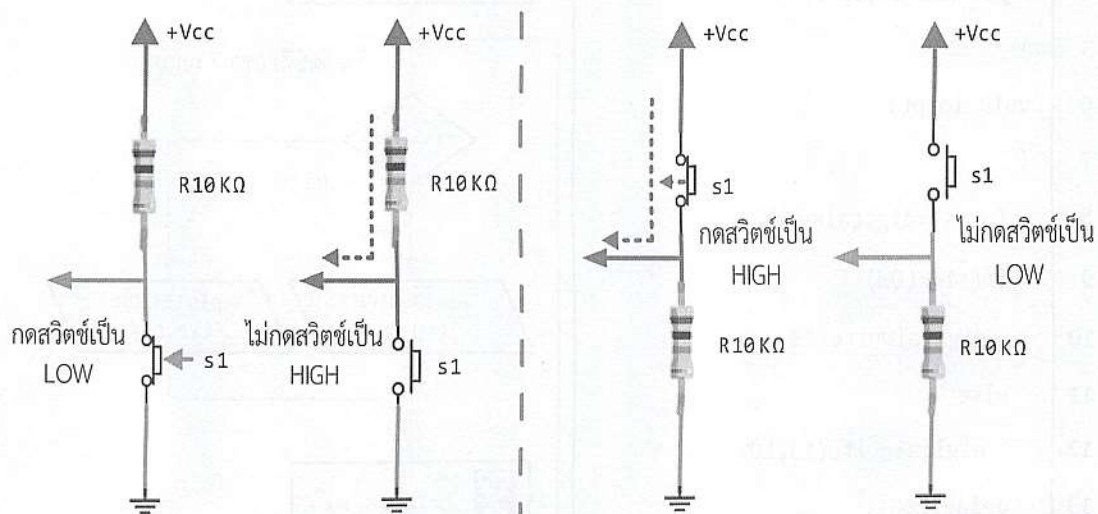
s1=digitalRead(7);	หมายถึง	อ่านค่าจากขา 7 ว่าเป็นลอจิก 0 หรือลอจิก 1
if(digitalRead(7)==HIGH)	หมายถึง	ถ้าขา 7 เป็น HIGH หรือลอจิก 1 ให้ทำคำสั่งในฟังก์ชัน if()



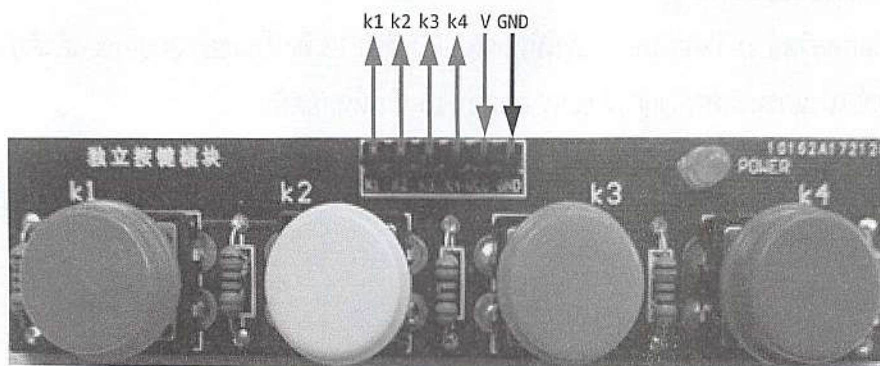


รูปที่ 1.22 การทำงานของวงจรสวิตช์

การต่อวงจรสวิตช์สามารถต่อได้ 2 แบบ คือ Active LOW และ Active HIGH หลักการทำงานของ Active LOW คือเมื่อกดสวิตช์จะให้เอาต์พุตเป็น LOW หรือ 0 เมื่อปล่อยสวิตช์จะเป็น HIGH หรือ 1 ดังรูปที่ 1.22 ส่วนวงจรสวิตช์แบบ Active HIGH จะทำงานตรงข้ามกัน คือเมื่อกดสวิตช์จะให้เอาต์พุตเป็น HIGH หรือ 1 เมื่อปล่อยสวิตช์จะเป็น LOW หรือ 0 เพื่อความสะดวกในการใช้งานสามารถใช้วงจรสวิตช์ซึ่งมีสวิตช์ 4 อินพุตดังรูปที่ 1.24



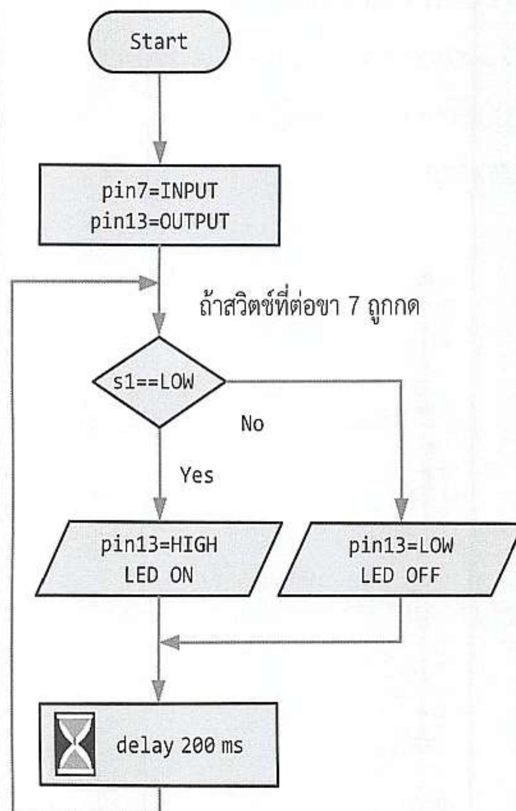
รูปที่ 1.23 วงจรสวิตช์แบบ Active LOW และ Active HIGH



รูปที่ 1.24 วงจรสวิตช์ 4 อินพุตแบบ Active Low

ตัวอย่างที่ 1.4 โปรแกรมรับข้อมูลจากสวิตช์

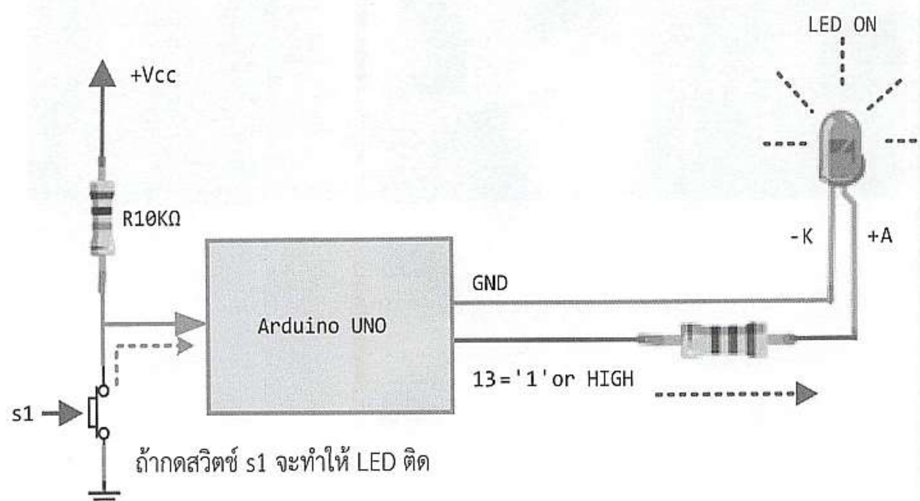
```
void setup()
{
  pinMode(7,INPUT);
  pinMode(13,OUTPUT);
}
void loop()
{
  int s1=digitalRead(7);
  if(s1==LOW)
    digitalWrite(13,HIGH);
  else
    digitalWrite(13,LOW);
  delay(200);
}
```



รูปที่ 1.25 โฟลว์ชาร์ตการรับข้อมูลจากสวิตช์

ผลการรันโปรแกรม

เมื่อกดสวิตช์ s1 ที่ต่อกับขา 7 ทำให้หลอด LED ที่ขา 13 ติดเป็นเวลา 200 ms แล้วดับ แต่ถ้าไม่มีการกดสวิตช์ โปรแกรมจะส่งสัญญาณ LOW ออกขา 13 ทำให้หลอดดับ



รูปที่ 1.26 การรับข้อมูลจากสวิตช์

ตัวอย่างที่ 1.5 โปรแกรมตรวจสอบพัลส์ขอบขาขึ้น การเปลี่ยนสัญญาณลอจิก 0 เป็นลอจิก 1

```
int state,pulse=1;                  //ประกาศตัวแปรเป็นชนิดจำนวนเต็ม
 void setup()
  {
   pinMode(7,INPUT);                //ให้ขา 7 เป็นอินพุต
   pinMode(13,OUTPUT);              //ให้ขา 13 เป็นเอาต์พุต
  }
void loop()
  { int s1=digitalRead(7);          //รับค่าอินพุตจากขา 7
   if(s1==0)   pulse=0;             //ถ้ากดสวิตช์ s1 ให้ pulse = 0
   if(s1==1 && pulse==0)            //ถ้าปล่อยสวิตช์และ pulse = 0
    {
     state=!state;                  //กลับสถานะตัวแปร state
     pulse=1;
    }
   if(state==1)                     //ถ้า state = 1
    digitalWrite(13,HIGH);
   else
    digitalWrite(13,LOW);
  }
```

ผลการรันโปรแกรม

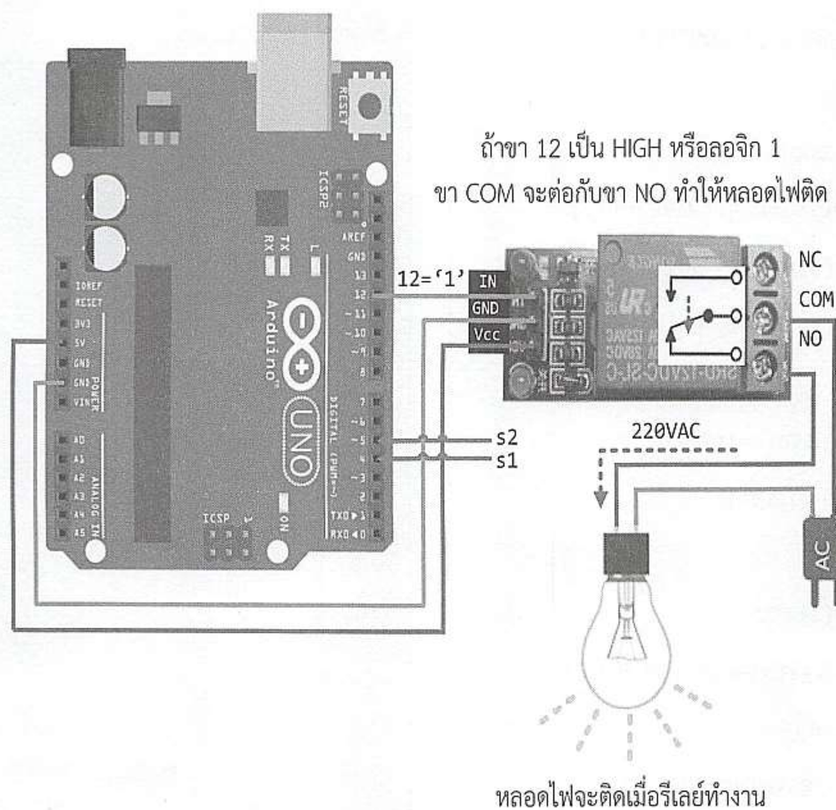
เมื่อกดสวิตช์ที่ขา 7 จะทำให้ตัวแปร s1 = 0 และตัวแปร pulse = 0 เมื่อปล่อยสวิตช์ ตัวแปร s1 = 1 และตัวแปร pulse = 0 เพื่อทำการตรวจสอบขอบขาขึ้นของสัญญาณพัลส์ ทำให้หลอด LED ที่ขา 13 จะติดดับตามการกดสวิตช์ โดยไม่ต้องใช้ฟังก์ชัน delay(); ในการหน่วงเวลา



รูปที่ 1.27 การตรวจสอบขอบขาขึ้นของสัญญาณพัลส์

1.9 การควบคุมรีเลย์

รีเลย์ (Relay) คือสวิตช์แม่เหล็กไฟฟ้า ทำหน้าที่ตัดหรือต่อวงจร เมื่อป้อนไฟเข้าขดลวดหรือส่งสัญญาณ HIGH หรือลอจิก 1 ออกที่ขา 12 เข้าขา IN จะทำให้รีเลย์ทำงาน ขาร่วมหรือขา COM (Common) ต่อกับขา NO (Normal Open) จะมีกระแสไหลในวงจรทำให้หลอดไฟ 220 โวลต์ติด และหากสัญญาณที่ขา 12 เป็น LOW จะทำให้รีเลย์ไม่ทำงานและทำให้หลอดไฟ 220 โวลต์ดับ



รูปที่ 1.28 การเชื่อมต่อกับรีเลย์

คุณสมบัติของรีเลย์ สามารถขับโหลดได้สูงสุด AC 250V/10A, DC 30V/10A แรงดันไฟฟ้าขณะทำงาน 5 โวลต์ และใช้กระแสไฟฟ้ากระตุ้นขดลวด 5 มิลลิแอมแปร์

ขาสัญญาณของรีเลย์

IN (Input) ขาสัญญาณควบคุมให้รีเลย์ทำงานหรือไม่ทำงาน

NO (Normal Open) ขาสัญญาณแบบปกติเปิด ซึ่งจะต่อกับโหลดหรืออุปกรณ์ที่ต้องการควบคุม

COM (Common) ขาร่วมต่อกับแหล่งจ่าย

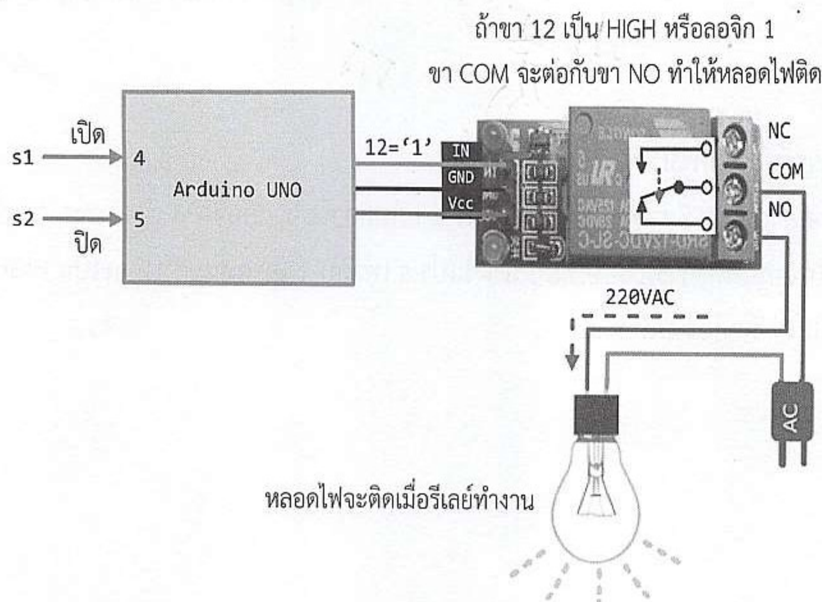
NC (Normal Close) ขาสัญญาณแบบปกติปิด จะต่อกับโหลดหรืออุปกรณ์ที่ต้องการทำงานแบบกลับสถานะ

ตัวอย่างที่ 1.6 โปรแกรมควบคุมรีเลย์ตามการกดสวิตช์

```
void setup()                    //กำหนดค่าเริ่มต้น
{
  pinMode(12,OUTPUT);           //กำหนดให้ขา 12 เป็นเอาต์พุต
  pinMode(4,INPUT);             //กำหนดขา 4 เป็นอินพุต
  pinMode(5,INPUT);             //กำหนดขา 5 เป็นอินพุต
}
void loop()
{
  if(digitalRead(4)==LOW)       //ถ้ากดสวิตช์ที่ขา 4
    digitalWrite(12,HIGH);      //ให้รีเลย์ที่ต่อกับขา 12 ทำงาน
  if(digitalRead(5)==LOW)       //ถ้ากดสวิตช์ที่ขา 5
    digitalWrite(12,LOW);       //ให้รีเลย์หยุดทำงาน ขา 12 = 0
  delay(200);                   //หน่วงเวลา 200 ms
}
```

ผลการรันโปรแกรม

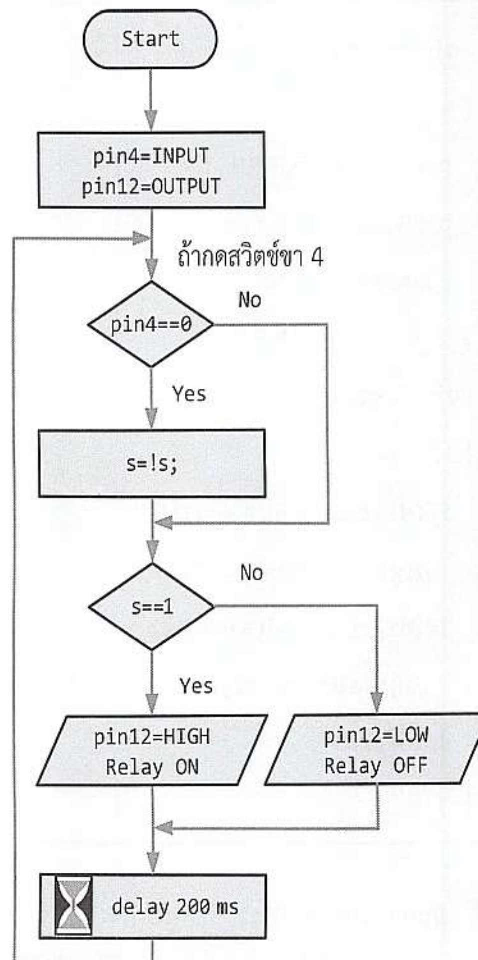
เมื่อกดสวิตช์ที่ขา 4 รีเลย์จะทำงาน ทำให้หลอดไฟ 220 โวลต์ติด และรีเลย์จะหยุดทำงานเมื่อกดสวิตช์ที่ขา 5



รูปที่ 1.29 การควบคุมรีเลย์ตามการกดสวิตช์ s1 และ s2

ตัวอย่างที่ 1.7 โปรแกรมปิด-เปิดรีเลย์โดยใช้สวิตช์แบบตัวเดียว (Toggle)

```
int s=0;
void setup()
{
  pinMode(4,INPUT);
  pinMode(12,OUTPUT);
}
void loop()
{
  if(digitalRead(4)==0)
    s=!s;
  if(s==1)
    digitalWrite(12,HIGH);
  else
    digitalWrite(12,LOW);
  delay(200);
}
```



รูปที่ 1.30 โฟลว์ชาร์ตควบคุมรีเลย์

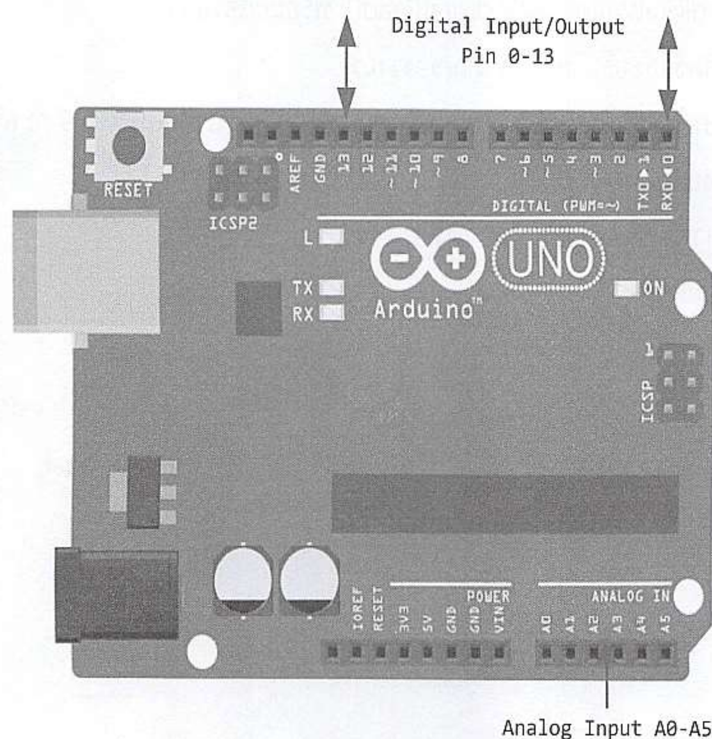
ผลการรันโปรแกรม

เมื่อกดสวิตช์ s1 ที่ขา 4 จะทำให้ตัวแปร s กลับสถานะจากลอจิก 0 เป็นลอจิก 1 หรือจากลอจิก 1 เป็นลอจิก 0 จากนั้นทำการเปรียบเทียบ ถ้าตัวแปร s เท่ากับ 1 เอาต์พุตขา 12 จะเป็น HIGH ทำให้รีเลย์ทำงาน แต่ถ้าไม่ใช่ รีเลย์จะไม่ทำงาน

1.10 สรุป

ไมโครคอนโทรลเลอร์ คือตัวควบคุมขนาดเล็กที่ใช้เทคโนโลยีการผลิตในระดับไมโคร มีหลากหลายตระกูล เช่น MCS-51 PIC AVR และ Arduino

Arduino เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิตในตระกูล AVR ที่ถูกออกแบบมาเพื่อให้ใช้งานง่าย สามารถเชื่อมต่อกับคอมพิวเตอร์ด้วยพอร์ต USB



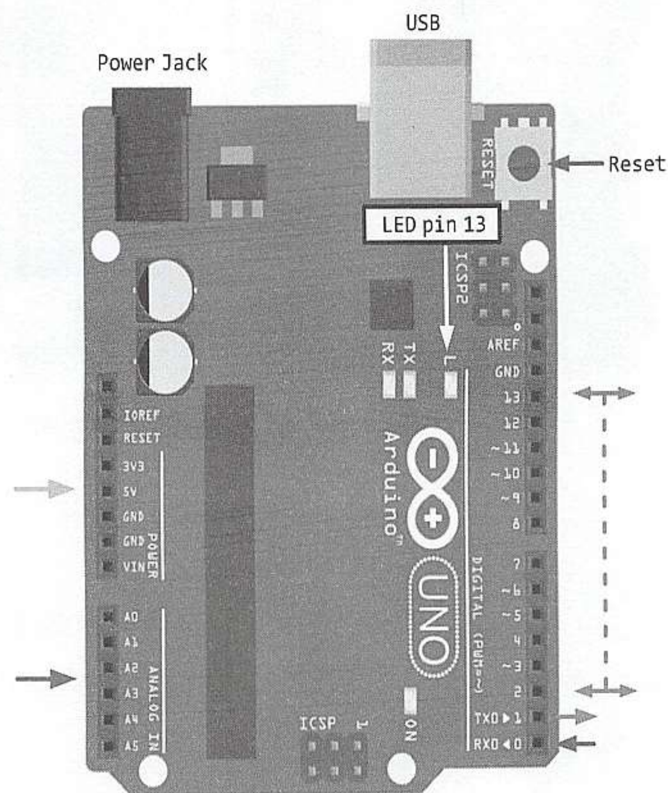
รูปที่ 1.31 ขาสัญญาณอินพุตและเอาต์พุต

Arduino UNO มีขา 0–13 เป็นขาดิจิทัลอินพุตและเอาต์พุต ในการเขียนโปรแกรมจะต้องมีการกำหนดขาสัญญาณให้เป็นอินพุตหรือเอาต์พุตโดยใช้คำสั่ง pinMode(); และถ้าต้องการส่งข้อมูลดิจิทัลออกต้องใช้คำสั่ง digitalWrtie(); ส่วนการรับข้อมูลเข้าจะใช้คำสั่ง digitalRead();

pinMode(13,OUTPUT);	หมายถึง กำหนดให้ขา 13 เป็นขาเอาต์พุตเพื่อส่งสัญญาณออกไปควบคุมอุปกรณ์
pinMode(3,INPUT);	หมายถึง กำหนดให้ขา 3 เป็นขาอินพุตเพื่อรับข้อมูลจากสวิตช์หรือเซนเซอร์
digitalWrite(13,HIGH);	หมายถึง ให้เอาต์พุตขา 13 เป็นลอจิก 1 หรือส่งสัญญาณไฟ 5 โวลต์ออกที่ขา 13
if(digitalRead(2)==LOW)	หมายถึง ถ้าค่าข้อมูลที่ขา 2 เป็น LOW ให้ทำคำสั่งในฟังก์ชัน if()

แบบฝึกหัดบทที่ 1

1. ไมโครคอนโทรลเลอร์คืออะไร
2. อธิบายฟังก์ชัน pinMode(); พร้อมยกตัวอย่าง
3. อธิบายฟังก์ชัน digitalWrite(); และ digitalRead(); พร้อมยกตัวอย่าง
4. ให้เขียนวงจรสวิตช์และอธิบายการทำงานของวงจร
5. ให้เขียนโปรแกรมไฟกะพริบที่ขา 8 โดยกะพริบทุก 1000 มิลลิวินาที และที่ขา 9 กะพริบทุก 500 มิลลิ-วินาที โดยใช้ฟังก์ชัน millis();
6. อธิบายขาสัญญาณของ Arduino Uno ดังรูป



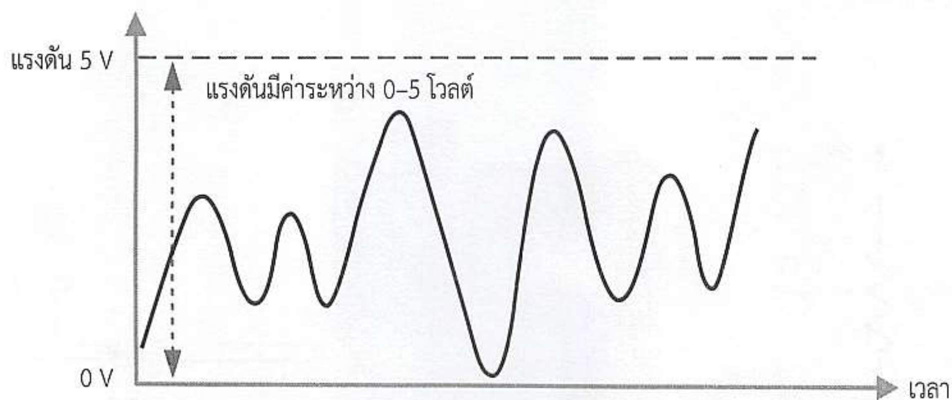
รูปที่ 1.32 อธิบายขาสัญญาณอินพุตและเอาต์พุต

บทที่

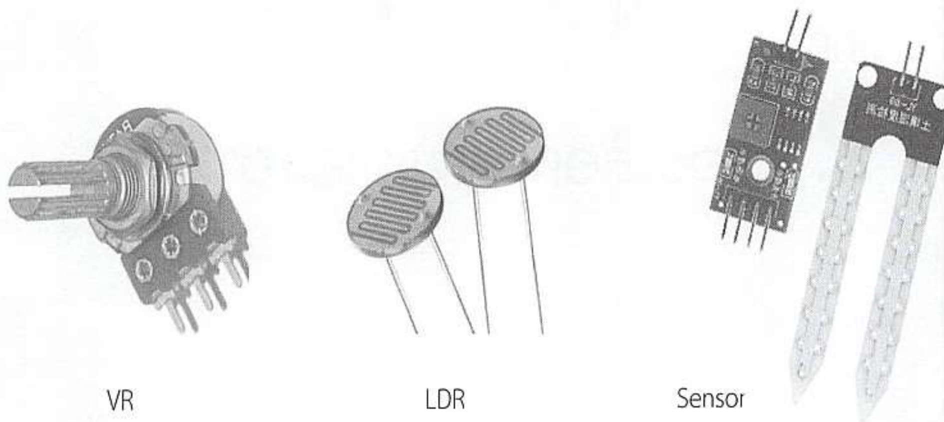
2 แอนะล็อกอินพุตและเอาต์พุต

แอนะล็อกอินพุต (Analog Input) คือสัญญาณอินพุตที่ปรับเปลี่ยนค่าได้ มีค่าระหว่าง 0–5 โวลต์ เช่น 0.5 2.4 หรือ 4.1 โวลต์ สามารถต่อเข้ากับขาแอนะล็อกอินพุตที่ขา A0–A5 ตัวอย่างอุปกรณ์แอนะล็อกอินพุต ได้แก่ ความต้านทานแบบปรับค่าได้ อุปกรณ์วัดค่าแสง และอุปกรณ์วัดความชื้นในดิน

แอนะล็อกเอาต์พุต (Analog Output) คือสัญญาณเอาต์พุตที่ปรับค่าได้ มีค่าระหว่าง 0–255 ซึ่งเป็นสัญญาณแอนะล็อกเอาต์พุตแบบ PWM มีความละเอียด 8 บิต ในการใช้งานสามารถต่อกับอุปกรณ์เอาต์พุตเพื่อใช้ปรับความสว่างของหลอดไฟหรือปรับความเร็วของมอเตอร์ได้



รูปที่ 2.1 สัญญาณแอนะล็อก

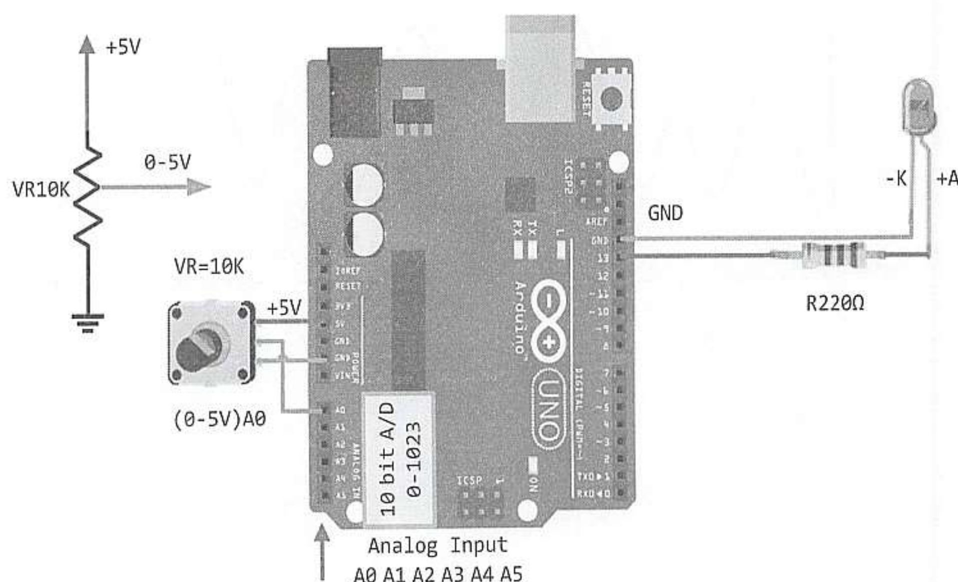


รูปที่ 2.2 ตัวอย่างอุปกรณ์แอนะล็อกอินพุต

2.1 ฟังก์ชัน analogRead();

Arduino UNO มีขาสัญญาณแอนะล็อกอินพุต 6 ขา คือ A0–A5 โดยแรงดันที่ป้อนเข้าขา A0–A5 มีค่าระหว่าง 0–5 โวลต์ แล้วผ่านวงจรแปลงสัญญาณแอนะล็อกให้เป็นสัญญาณดิจิทัลซึ่งเป็นวงจรที่มีอยู่ภายในไมโครคอนโทรลเลอร์ สัญญาณดิจิทัลที่ได้จะมีค่า 0–1023 มีความละเอียดขนาด 10 บิต (10-bit A/D) ในการอ่านค่าแอนะล็อกจะใช้ฟังก์ชัน analogRead();

analogRead(); คือฟังก์ชันสำหรับอ่านข้อมูลแอนะล็อกอินพุตจากขา A0–A5 โดยค่าที่อ่านได้จะอยู่ในช่วง 0–1023



รูปที่ 2.3 วงจรการอ่านค่าความต้านทานแบบปรับค่าได้

ตัวอย่าง

int VR=analogRead(A0);

หมายถึง ให้อ่านค่าข้อมูลจากขาแอนะล็อก A0 มาเก็บไว้ในตัวแปร VR ซึ่งเป็นตัวแปรแบบเลขจำนวนเต็ม 16 บิต โดยสัญญาณจากค่าความต้านทานแบบปรับค่าได้มีค่าอยู่ระหว่าง 0–5 โวลต์ ซึ่งจะถูกแปลงให้เป็นสัญญาณดิจิทัลที่มีค่า 0–1023

ตัวต้านทานแบบปรับค่าได้ (Variable Resistor : VR) คือตัวต้านทานที่สามารถปรับค่าความต้านทานและแรงดันไฟฟ้าเอาต์พุต มีขาสัญญาณ 3 ขา คือ ขาสัญญาณไฟ ขากราวด์ และขากลางหรือขาสัญญาณแรงดันไฟฟ้าเอาต์พุต

2.2 ฟังก์ชัน Serial

Serial เป็นฟังก์ชันเกี่ยวกับการรับและส่งข้อมูลแบบอนุกรม เพื่อแสดงผลของข้อมูลบนจอภาพโดยใช้เมนู Serial Monitor

Serial.begin(); กำหนดอัตราการรับและส่งข้อมูลแบบอนุกรม มีหน่วยเป็นบิตต่อวินาที เช่น 4800 9600 14400 19200 หรือ 38400

Serial.print(); เป็นฟังก์ชันที่ใช้แสดงผลหรือพิมพ์ข้อมูลที่รับมาจากบอร์ด Arduino โดยการรับและส่งข้อมูลแบบอนุกรม

Serial.println(); เป็นฟังก์ชันที่ใช้แสดงผลหรือพิมพ์ข้อมูลโดยจะขึ้นบรรทัดใหม่ทุกครั้ง

ตัวอย่าง

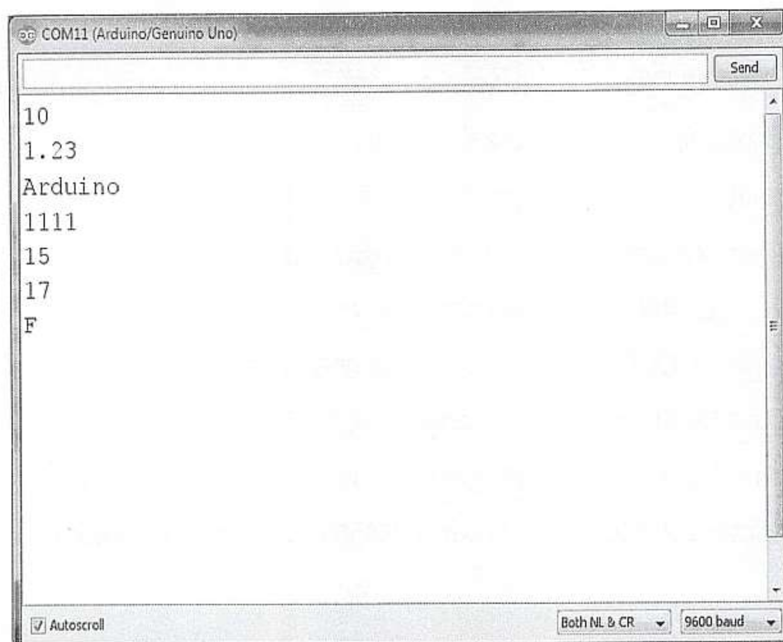
Serial.begin(9600);	กำหนดอัตราการรับและส่งข้อมูลแบบอนุกรม 9600 บิตต่อวินาที ซึ่งต้องกำหนดใน void setup()
Serial.begin(19200);	กำหนดอัตราการรับและส่งข้อมูลแบบอนุกรม 19200 บิตต่อวินาที
Serial.print(10);	แสดงตัวเลข 10
Serial.print('A');	แสดงตัวอักษร A
Serial.print("Arduino");	แสดงข้อความ Arduino
Serial.print(15, BIN);	แสดงเลขฐานสองของ 15 คือ 1111
Serial.print(15, OCT);	แสดงเลขฐานแปดของ 15 คือ 17
Serial.print(15, DEC);	แสดงเลขฐานสิบของ 15 คือ 15
Serial.print(15, HEX);	แสดงเลขฐานสิบหกของ 15 คือ F
Serial.println(1.23456, 2);	แสดงเลข 1.23456 เป็นทศนิยม 2 ตำแหน่งคือ 1.23 และขึ้นบรรทัดใหม่

ตัวอย่างที่ 2.1 โปรแกรมแสดงข้อมูลผ่าน Serial Monitor

```
int a=15;                          //ประกาศตัวแปร a เป็นชนิดจำนวนเต็มเท่ากับ 15
void setup()
{
  Serial.begin(9600);              //กำหนดอัตราการสื่อสารข้อมูล
}
void loop()
{
    Serial.println(10);            //แสดงเลข 10 แล้วขึ้นบรรทัดใหม่
    Serial.println(1.234);         //แสดงเลข 1.23 แล้วขึ้นบรรทัดใหม่
    Serial.println(“Arduino”);     //แสดงข้อความ Arduino แล้วขึ้นบรรทัดใหม่
    Serial.println(a,BIN);         //แสดงตัวแปร a เป็นเลขฐานสองแล้วขึ้นบรรทัดใหม่
    Serial.println(a,DEC);         //แสดงตัวแปร a เป็นเลขฐานสิบแล้วขึ้นบรรทัดใหม่
    Serial.println(a,OCT);         //แสดงตัวแปร a เป็นเลขฐานแปดแล้วขึ้นบรรทัดใหม่
    Serial.println(a,HEX);         //แสดงตัวแปร a เป็นเลขฐานสิบหกแล้วขึ้นบรรทัดใหม่
}
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงข้อมูลในรูปแบบต่าง ๆ ออกทาง Serial Monitor



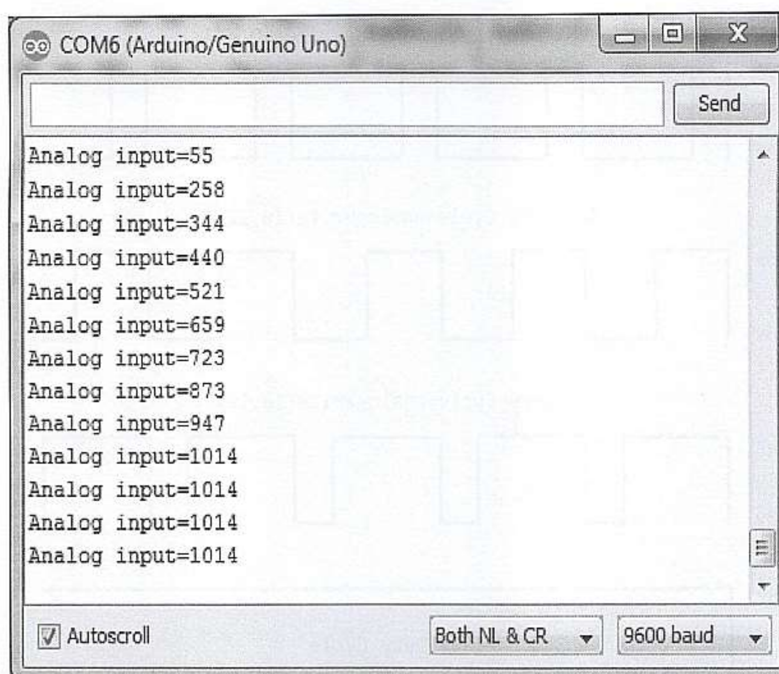
รูปที่ 2.4 การแสดงข้อมูลของ Serial Monitor

ตัวอย่างที่ 2.2 โปรแกรมรับค่าสัญญาณแอนะล็อกจากความต้านทานปรับค่าได้

```
void setup()
{
  Serial.begin(9600);              //กำหนดอัตราการสื่อสารข้อมูล
}
void loop()
  {
    int VR=analogRead(A0);         //อ่านค่าจากขา A0
    Serial.print("Analog input="); //แสดงข้อความ Analog input=
    Serial.println(VR);            //แสดงค่าที่อ่านได้จากขา A0 แล้วขึ้นบรรทัดใหม่
    delay(500);
  }
```

ผลการรันโปรแกรม

โปรแกรมวนรอบอ่านค่าสัญญาณแอนะล็อกจากขา A0 ทุก ๆ 500 มิลลิวินาที แล้วนำมาแสดงผล โดยค่าที่อ่านได้อยู่ในช่วง 0–1023 โดยจะแปรเปลี่ยนตามการปรับค่าความต้านทาน



รูปที่ 2.5 การอ่านค่าสัญญาณแอนะล็อกจากความต้านทานปรับค่าได้

2.3 ฟังก์ชัน analogWrite();

Arduino UNO มีขาเอาต์พุตแอนะล็อกแบบ PWM 6 ขา คือ 3 5 6 9 10 และ 11 เป็นสัญญาณ PWM ขนาด 8 บิต มีค่า 0–255 โดย PWM คือสัญญาณที่มีการปรับความกว้างของสัญญาณพัลส์ตามสัดส่วนที่กำหนดเพื่อปรับแรงดันเฉลี่ยของเอาต์พุต สามารถใช้ในการควบคุมความสว่างของหลอดไฟหรือควบคุมความเร็วของมอเตอร์ โดยการส่งสัญญาณแอนะล็อกจะใช้ฟังก์ชัน analogWrite();

analogWrite(); คือฟังก์ชันส่งข้อมูลแอนะล็อกแบบ PWM ซึ่งมีค่าอยู่ในช่วง 0–255

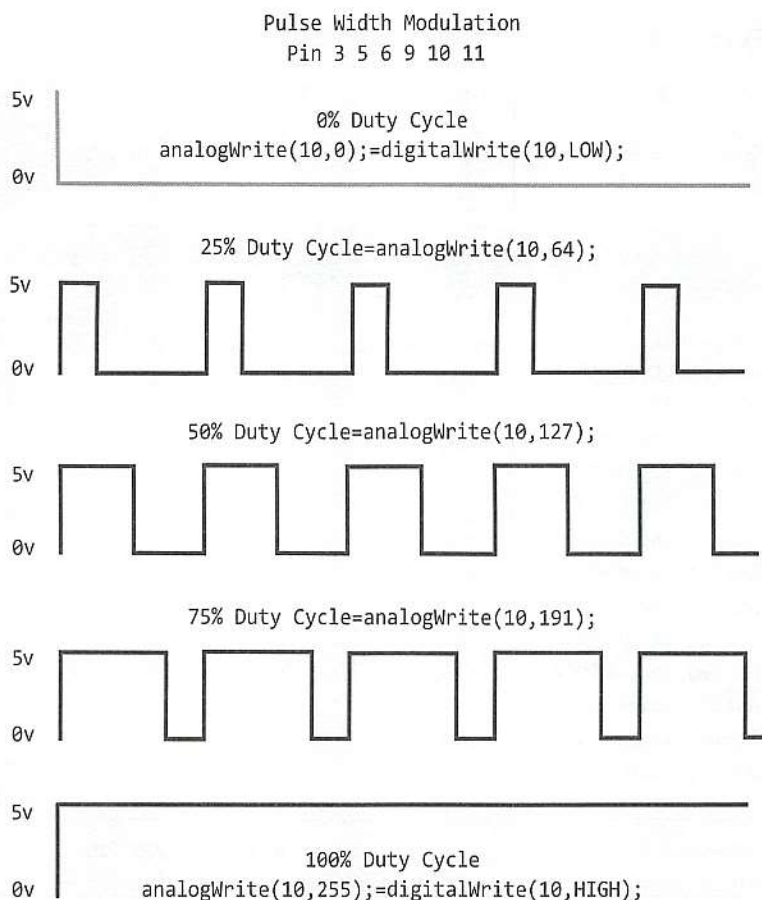
ตัวอย่าง

```
analogWrite(10,64);
```

หมายถึง ให้ส่งสัญญาณพัลส์ที่มีความกว้างของสัญญาณพัลส์ 25% (64 = 25%) ออกขา 10

```
analogWrite(10,255);
```

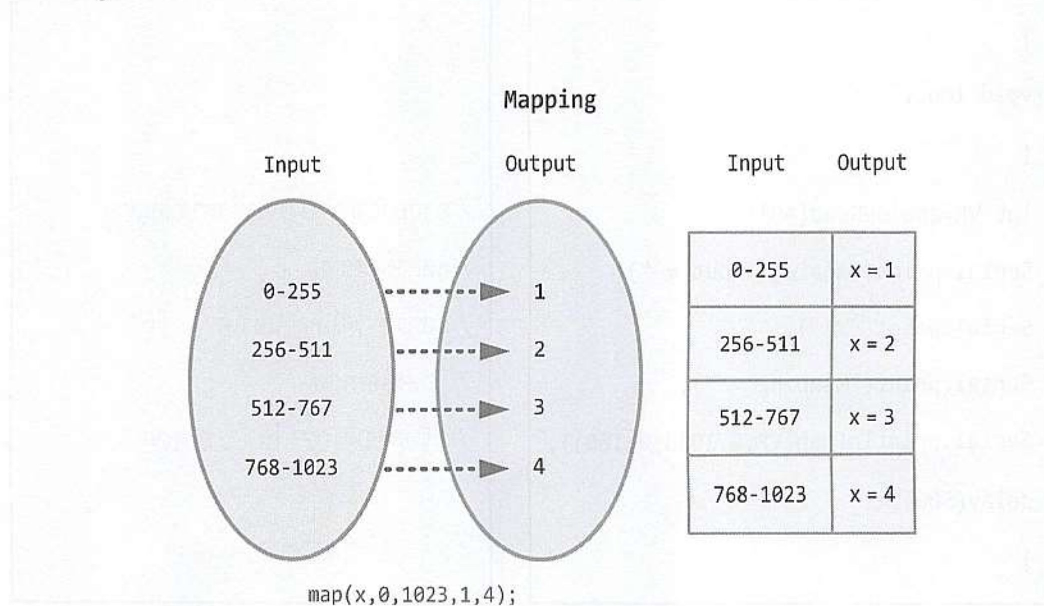
หมายถึง ให้ส่งสัญญาณพัลส์ที่มีความกว้างของสัญญาณพัลส์ 100% (255 = 100%) ซึ่งเป็นค่าสูงสุดออกขา 10 ซึ่งมีค่าเท่ากับ digitalWrite(10,HIGH);



รูปที่ 2.6 สัญญาณ PWM

2.4 ฟังก์ชัน map();

map(); เป็นฟังก์ชันที่ใช้ในการปรับค่าตัวเลข จากค่าตัวเลขช่วงจำนวนหนึ่งให้เป็นอีกช่วงจำนวนหนึ่ง เพื่อปรับค่าตัวแปรให้เหมาะสมในการใช้งาน เช่น ค่าที่อ่านได้มีค่า 0–1023 แต่ต้องการปรับให้เป็น 1 ถึง 4 แสดงดังรูปที่ 2.7



รูปที่ 2.7 การทำงานของฟังก์ชัน map();

ตัวอย่าง

x=map(x,0,1023,1,4);

หมายถึง การแปลงค่าของตัวแปร x จาก 0–1023 ให้อยู่ในช่วง 1–4 แล้วเก็บไว้ในตัวแปร x

VR=map(VR,0,1023,0,255);

หมายถึง การแปลงค่าของตัวแปร VR จาก 0–1023 ให้อยู่ในช่วง 0–255 แล้วเก็บไว้ในตัวแปร VR ซึ่งช่วงตัวเลข 0–255 สามารถใช้ในการส่งค่าเอาต์พุตแบบ PWM

ตัวอย่างที่ 2.3 โปรแกรมรับค่าสัญญาณแอนะล็อก 0–1023 แปลงค่าเป็น 0–100

```
void setup()                              //กำหนดอัตราการสื่อสารข้อมูล
{ Serial.begin(9600);
}
void loop()
{
int VR=analogRead(A0);                    //อ่านค่าจากขา A0 มาสื่อสารข้อมูล
Serial.print("Analog input = ");          //แสดงข้อความ
Serial.print( VR );                       //แสดงค่า VR ที่อ่านค่าได้
Serial.print("Mapping = ");               //แสดงข้อความ
Serial.println(map(VR,0,1023,0,100));     //ปรับค่า 0–1023 ให้เป็น 0–100
delay(500);
}
```

ผลการรันโปรแกรม

โปรแกรมอ่านค่าสัญญาณแอนะล็อกอินพุตจากขา A0 ซึ่งมีค่าระหว่าง 0–1023 แล้วนำมาเก็บไว้ในตัวแปร VR จากนั้นทำการปรับค่าตามสัดส่วนให้อยู่ในช่วง 0–100 แล้วแสดงผลบนจอภาพ

```
COM11 (Arduino/Genuino Uno)                                   Send
Analog input = 0Mapping = 0
Analog input = 0Mapping = 0
Analog input = 110Mapping = 10
Analog input = 439Mapping = 42
Analog input = 721Mapping = 70
Analog input = 918Mapping = 89
Analog input = 1023Mapping = 100
Analog input = 590Mapping = 57
Analog input = 14Mapping = 1
Analog input = 241Mapping = 23
Analog input = 429Mapping = 41
Autoscroll                                 Both NL & CR   9600 baud
```

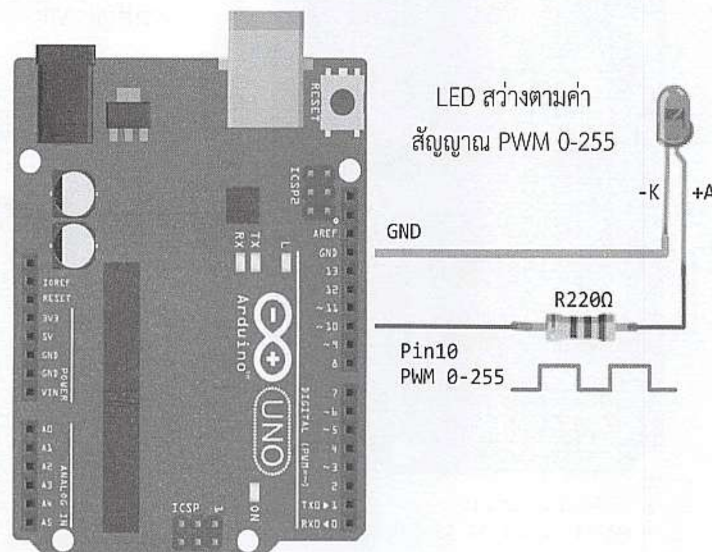
รูปที่ 2.8 การอ่านค่าสัญญาณแอนะล็อกผ่านฟังก์ชัน map();

ตัวอย่างที่ 2.4 โปรแกรมปรับความสว่างของหลอด LED

```
int P;
void setup()
{ pinMode(10,OUTPUT);           //ให้ขา 10 เป็นเอาต์พุต
}
void loop()
  { P=P+5;                      //เพิ่มค่า P ครั้งละ 5
    analogWrite(10,P);          //ส่งสัญญาณ PWM 0–255 ออกขา 10
    delay(50);
    if(P>255)                   //ถ้า P มากกว่า 255
    { P=0;                      //ให้ P เท่ากับ 0
      analogWrite(10,0);        //ให้ขา 10 เป็น 0 หลอดไฟดับ
    }
  }
```

ผลการรันโปรแกรม

โปรแกรมจะวนรอบส่งสัญญาณ PWM ออกขา 10 ด้วยคำสั่ง analogWrite(10,P); โดยค่าของสัญญาณ PWM ที่ส่งออกจะเริ่มจาก 0 แล้วเพิ่มขึ้นทีละ 5 จนถึง 255 ทำให้หลอดแสดงผล LED ค่อย ๆ สว่างขึ้น



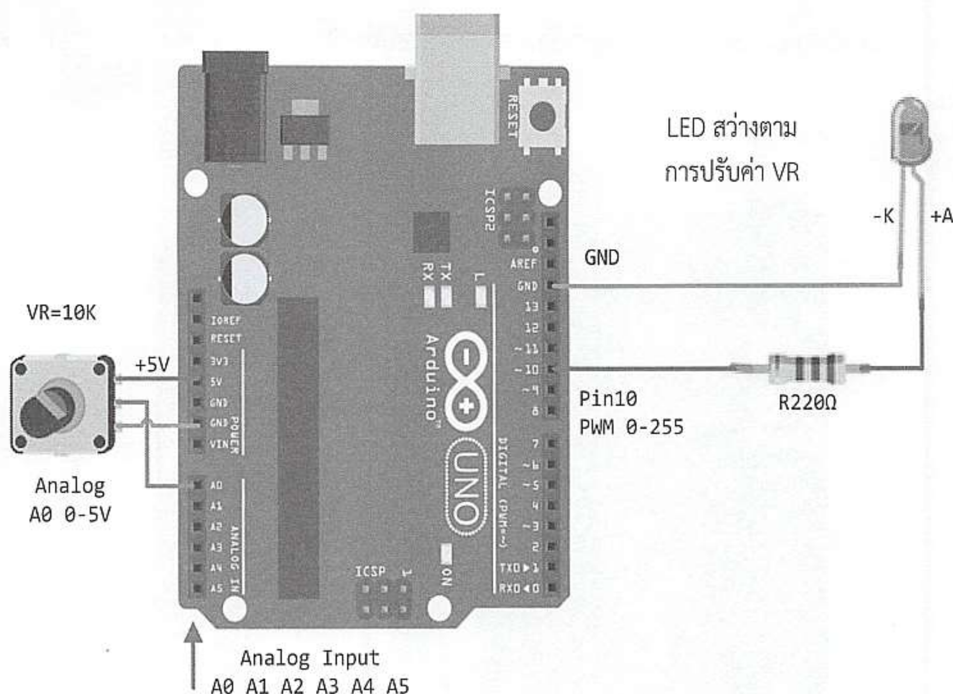
รูปที่ 2.9 วงจรการส่งสัญญาณ PWM ออกขา 10

ตัวอย่างที่ 2.5 โปรแกรมปรับความสว่างของหลอด LED ตามความต้านทานแบบปรับค่าได้

```
void setup()
{
pinMode(10,OUTPUT);                //ให้ขา 10 เป็นเอาต์พุต
}
void loop()
{
  int VR=analogRead(A0);           //อ่านค่าจากขา A0
  VR=map(VR,0,1023,0,255);         //แปลงค่า 0-1023 ให้เป็น 0-255
  analogWrite(10,VR);              //ส่งสัญญาณ PWM 0-255 ออกขา 10
}
```

ผลการรันโปรแกรม

โปรแกรมจะวนรอบรับข้อมูลจากขา A0 ที่มีค่า 0–1023 แล้วแปลงค่าให้เป็น 0–255 จากนั้นส่งสัญญาณ PWM ออกขา 10 ด้วยคำสั่ง analogWrite(10,VR); โดยค่าของสัญญาณ PWM ที่ส่งออกอยู่ในช่วง 0–255 ทำให้หลอด LED ที่ต่อกับขา 10 สว่างตามการปรับค่า VR

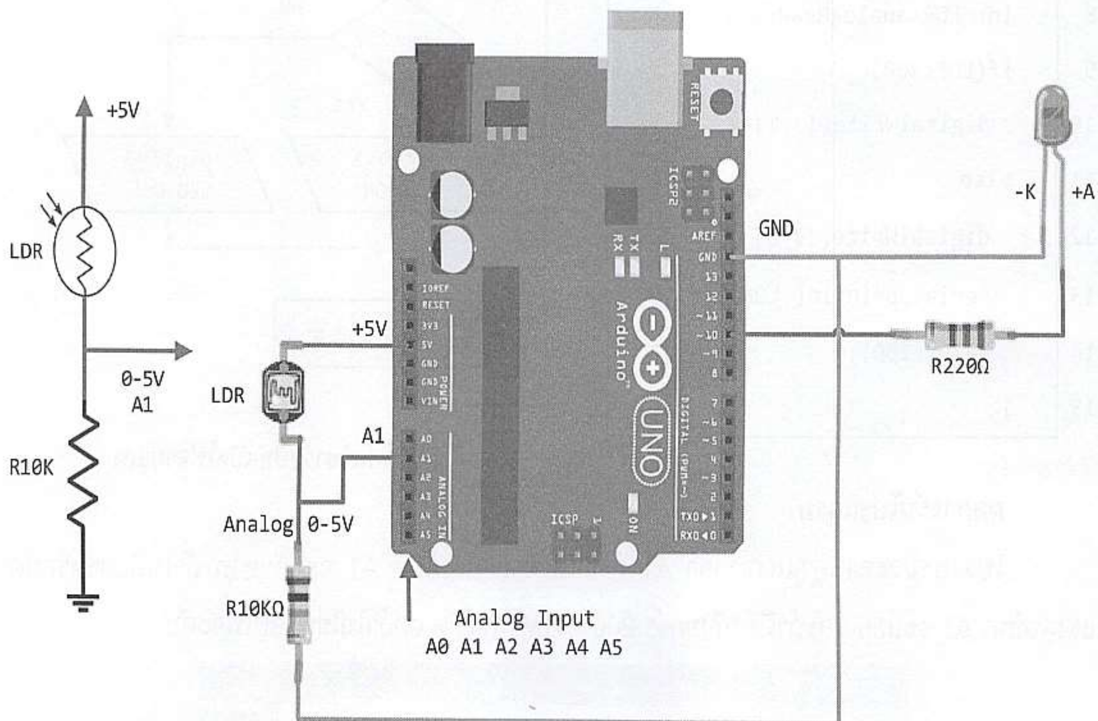


รูปที่ 2.10 วงจรปรับความสว่างหลอด LED

2.5 ตัวต้านทานแบบ LDR

LDR (Light Dependent Resistor) คือความต้านทานที่มีค่าเปลี่ยนไปตามแสงที่ตกกระทบ การทำงานของ LDR หากมีแสงตกกระทบจะทำให้ความต้านทานมีค่าลดลงเหลือประมาณหลักร้อยโอห์ม หากไม่มีแสง ความต้านทานจะมากขึ้นประมาณหลักกิโลโอห์ม ส่วนมากจะประยุกต์ใช้งานเกี่ยวกับแสง เช่น ตรวจวัดความเข้มของแสง หาทิศทางของแสง หรือตรวจสอบเวลากลางวัน-กลางคืน

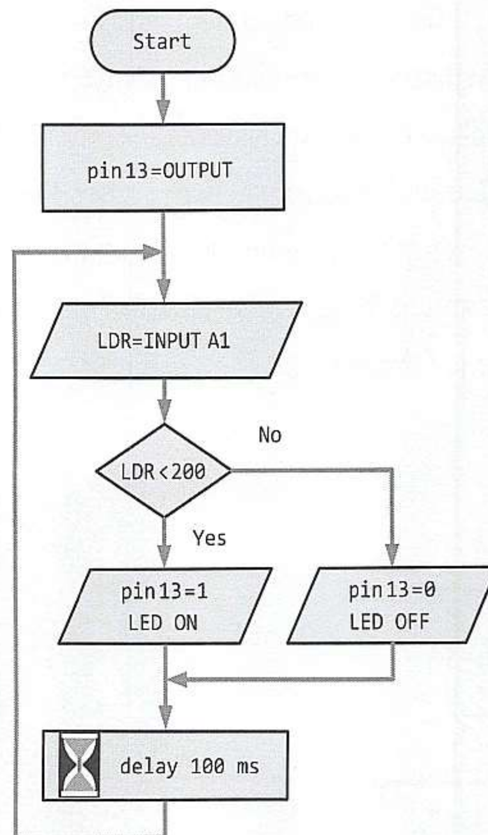
จากรูปที่ 2.11 เป็นการต่อวงจร LDR เข้าที่ขา A1 เมื่อมีแสงตกกระทบจะทำให้ความต้านทานของ LDR ลดลงและทำให้มีแรงดันเอาต์พุตเพิ่มขึ้น ซึ่งแรงดันเอาต์พุตจะแปรเปลี่ยนไปตามความเข้มของแสงที่ตกกระทบตัวต้านทาน LDR



รูปที่ 2.11 วงจรตรวจวัดแสงด้วย LDR

ตัวอย่างที่ 2.6 โปรแกรมปิด-เปิดไฟตามความสว่างของแสง

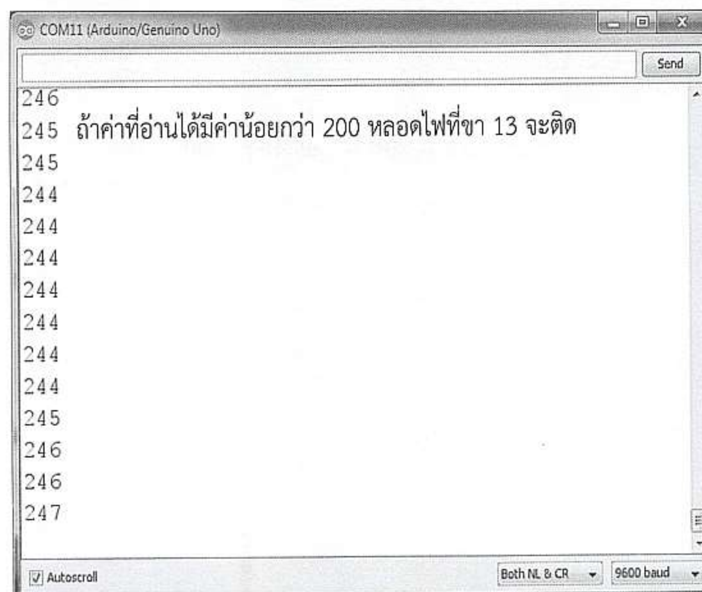
```
void setup()
{
Serial.begin(9600);
pinMode(13,OUTPUT);
}
void loop()
  {
  int LDR=analogRead(A1);
  if(LDR<200)
    digitalWrite(13,1);
  else
    digitalWrite(13,0);
    Serial.println( LDR );
    delay(100);
  }
```



รูปที่ 2.12 โฟลว์ชาร์ตปิด-เปิดไฟตามแสง

ผลการรันโปรแกรม

โปรแกรมจะอ่านค่าแสงจากขา A1 ถ้ามีแสงมาก แรงดันที่ A1 จะมีค่ามาก ถ้าไม่มีแสงหรือมืด แรงดันที่ขา A1 จะน้อย และถ้ามีค่าน้อยกว่า 200 หลอดไฟจะติด แต่ถ้าไม่ใช่หลอดไฟจะดับ



รูปที่ 2.13 ค่าแสงที่อ่านได้จาก Serial Monitor

2.6 การวัดความชื้นในดิน

การวัดความชื้นในดินจะมีเซนเซอร์หรือตัววัดความชื้นสำหรับเสียบลงไปในดิน แล้วผ่านวงจรขยายสัญญาณและปรับค่าแรงดัน ซึ่งค่าที่ได้เป็นแรงดันขนาด 0–5 โวลต์ตามระดับความชื้นในดิน ขาสัญญาณของวงจรวัดความชื้นในดินมีดังนี้

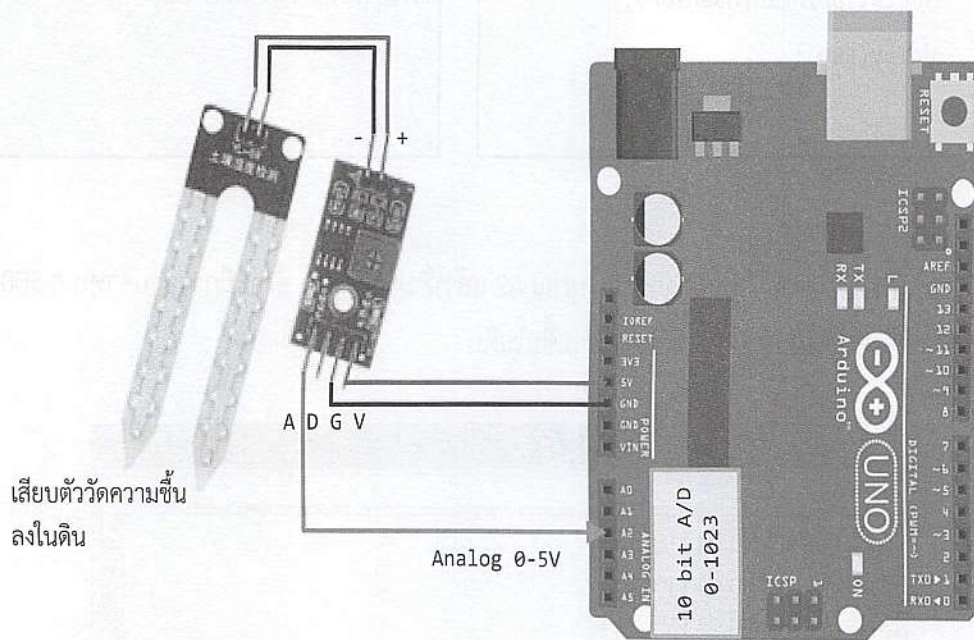
ขา + – ต่อเข้ากับตัววัดความชื้น

ขา A คือขาแอนะล็อกมีค่า 0–5 โวลต์ ต่อเข้ากับขา A0–A5 ของไมโครคอนโทรลเลอร์

ขา D คือขาสัญญาณเอาต์พุตดิจิทัล สามารถปรับค่าได้ที่ VR

ขา G ต่อกับขากราวด์

ขา V ต่อกับไฟ 5 โวลต์



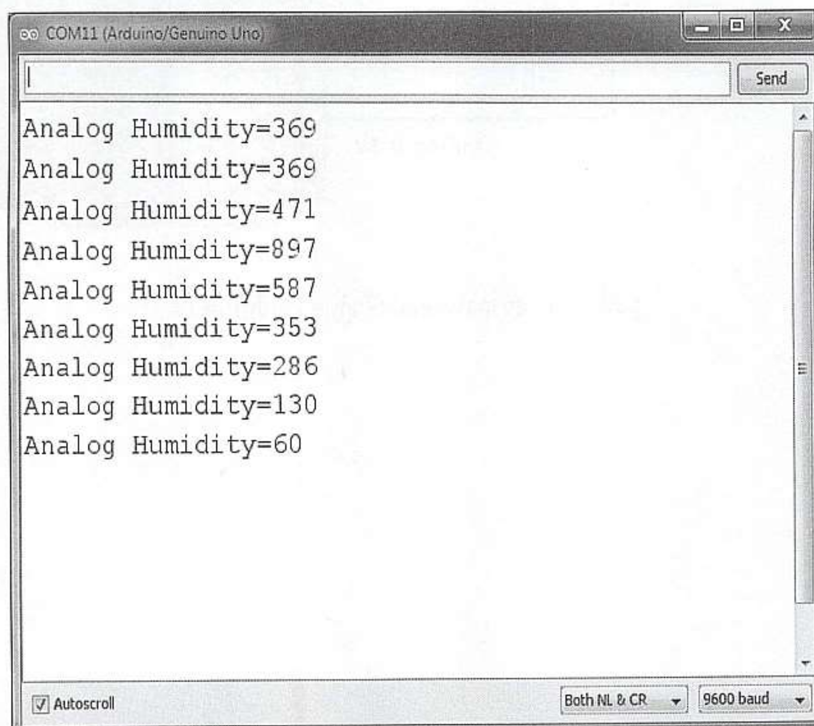
รูปที่ 2.14 วงจรการเชื่อมต่อตัววัดความชื้นในดิน

ตัวอย่างที่ 2.7 โปรแกรมอ่านค่าความชื้นในดิน

```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
    int HSensor=analogRead(A2);          //อ่านค่าความชื้นในดินจากขา A2
    Serial.print("Analog Humidity=");    //แสดงข้อความ
    Serial.println(HSensor);             //แสดงค่าความชื้นที่อ่านได้
    delay(500);
}
```

ผลการรันโปรแกรม

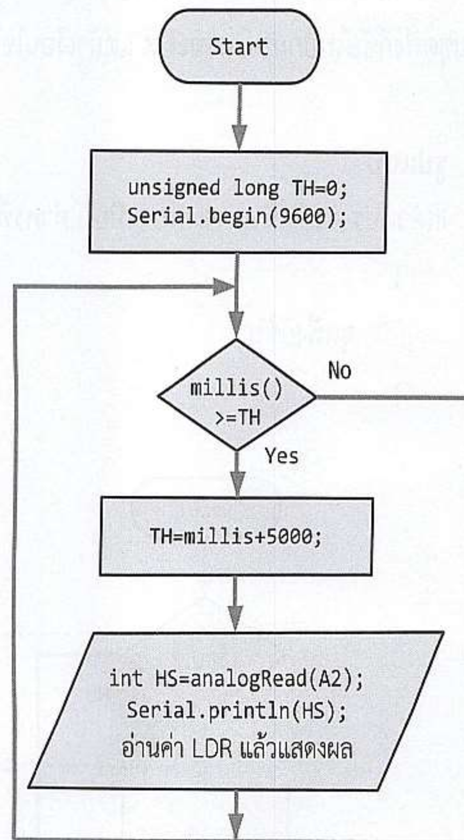
โปรแกรมอ่านค่าความชื้นจากขาสัญญาณ A2 แล้วนำมาแสดงผล ซึ่งจะมีการอ่านค่าทุก ๆ 500 ms โดยค่าที่อ่านได้อยู่ในช่วง 0–1023 ตามค่าความชื้นในดิน



รูปที่ 2.15 การอ่านค่าความชื้นในดิน

ตัวอย่างที่ 2.8 โปรแกรมอ่านค่าความชื้นในดินทุก 5 วินาทีโดยใช้ฟังก์ชัน millis();

```
unsigned long TH;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  if(millis()>=TH)
  {
    TH=millis()+5000;
    int HS=analogRead(A2);
    Serial.println(HS);
  }
}
```



รูปที่ 2.16 โฟลว์ชาร์ตอ่านค่าความชื้นในดินทุก 5 วินาที

ผลการรันโปรแกรม

โปรแกรมจะวนรอบอ่านค่าความชื้นในดินที่ขา A2 ทุก 5000 ms หรือ 5 วินาที โดยการตรวจสอบค่าของฟังก์ชัน millis(); ซึ่งจะนับอัตโนมัติ ทำให้ไมโครคอนโทรลเลอร์สามารถทำงานอื่นได้โดยไม่ติดอยู่ในฟังก์ชัน delay();

2.7 ฟังก์ชัน if() และ if_else

ฟังก์ชัน if() ทำหน้าที่เปรียบเทียบข้อมูล ถ้าเงื่อนไขในการเปรียบเทียบเป็นจริง โปรแกรมจะดำเนินการตามชุดฟังก์ชันในปีกกาเปิดและปิด แต่ถ้าเงื่อนไขไม่เป็นจริงหรือเป็นเท็จจะจบการทำงาน

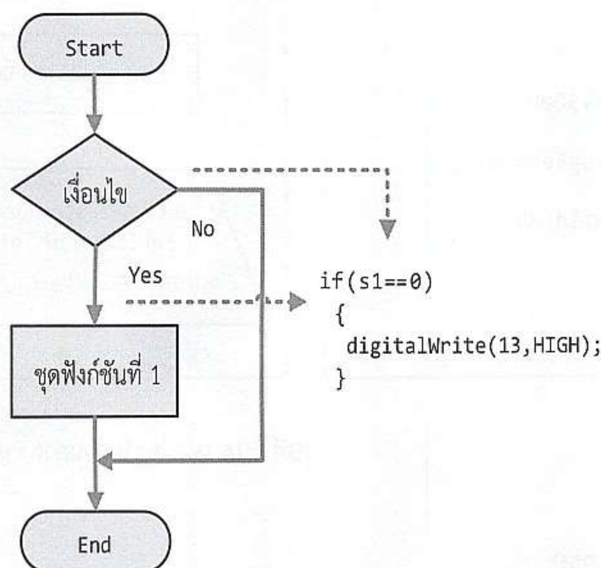
รูปแบบ

if(ตัวแปร ตัวดำเนินการเปรียบเทียบ ค่าคงที่หรือตัวแปร)

{

ชุดฟังก์ชัน

}



รูปที่ 2.17 การทำงานของฟังก์ชัน if()

ตัวอย่างการทำงานของฟังก์ชัน if()

```
if(s1==0)                    //ถ้า s1 เท่ากับ 0
{
    digitalWrite(13,HIGH);   //ให้ขา 13 เป็น HIGH
}
```

หมายถึง ถ้าตัวแปร s1 เท่ากับ 0 ให้ขา 13 เป็น HIGH หรือลอจิก 1 เครื่องหมายปีกกาเปิดและปิดในฟังก์ชัน if() ถ้าไม่ใส่จะทำฟังก์ชันเดียวเท่านั้น และฟังก์ชัน if() จะไม่มีเครื่องหมายเซมิโคลอน (;) ปิดท้าย ซึ่งแทนความหมายว่ายังไม่จบในบรรทัดนั้น ต้องทำคำสั่งระหว่างปีกกาเปิดและปิด การใช้ฟังก์ชัน if() จะใช้ร่วมกับตัวดำเนินการเปรียบเทียบและลอจิกดังนี้

1) ตัวดำเนินการเปรียบเทียบ

ใช้ในการเปรียบเทียบข้อมูลระหว่างตัวแปรกับตัวแปร หรือตัวแปรกับค่าคงที่ เพื่อเป็นเงื่อนไขในการใช้งานร่วมกับฟังก์ชัน if() ตัวดำเนินการเปรียบเทียบแสดงดังตารางที่ 2.1

ตารางที่ 2.1 ตัวดำเนินการเปรียบเทียบ

สัญลักษณ์	ความหมาย
>	มากกว่า
>=	มากกว่าหรือเท่ากับ
<	น้อยกว่า
<=	น้อยกว่าหรือเท่ากับ
==	เท่ากับหรือเท่ากัน
!=	ไม่เท่ากับหรือไม่เท่ากัน

ตัวอย่างการใช้งานตัวดำเนินการเปรียบเทียบ

```
if(s1==LOW)
digitalWrite(13,0);
```

หมายถึง ถ้าตัวแปร s1 เท่ากับ LOW หรือลอจิก 0 ให้ขา 13 เป็นลอจิก 0 หรือ LOW

2) ตัวดำเนินการลอจิก

ใช้งานร่วมกับฟังก์ชัน if() เพื่อเปรียบเทียบมากกว่าหนึ่งเงื่อนไข เช่น กรณีกดสวิตช์ทั้งสองตัว หรือกดสวิตช์ตัวใดตัวหนึ่ง หรือใช้ในการกลับสถานะของข้อมูล

ตารางที่ 2.2 ตัวดำเนินการลอจิก

สัญลักษณ์	ความหมาย	
&&	AND	และ
||	OR	หรือ
!	NOT	ไม่

ตัวอย่างการใช้งานตัวดำเนินการลอจิก

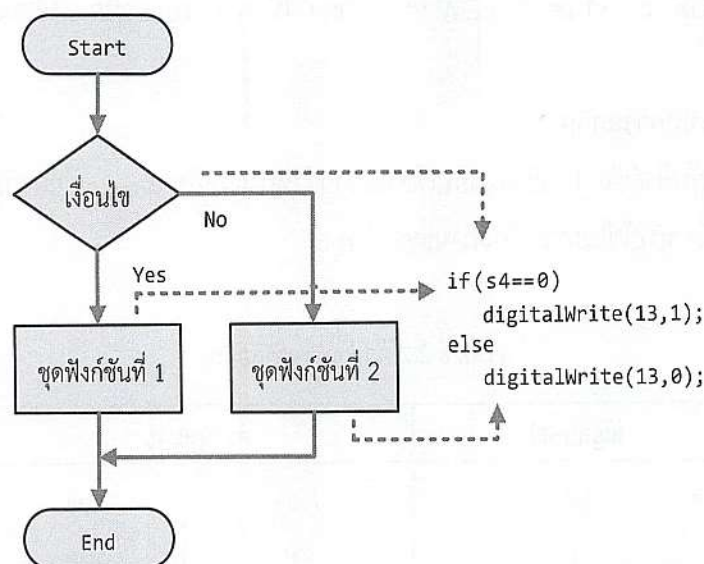
```
if(s1==LOW && s2==LOW)
digitalWrite(13,0);
```

หมายถึง ถ้า s1 และ s2 เท่ากับ LOW ให้ขา 13 เป็นลอจิก 0

ฟังก์ชัน if_else สองทางเลือกต้องใช้ร่วมกับ else การทำงานถ้าเงื่อนไขในการเปรียบเทียบเป็นจริงจะทำชุดฟังก์ชันที่ 1 แต่ถ้าเงื่อนไขเป็นเท็จจะทำชุดฟังก์ชันที่ 2

รูปแบบ

```
if(ตัวแปร ตัวดำเนินการเปรียบเทียบ ค่าคงที่หรือตัวแปร)
    {
        ชุดฟังก์ชันที่ 1
    }
else
    {
        ชุดฟังก์ชันที่ 2
    }
```



รูปที่ 2.18 โฟลว์ชาร์ตการทำงานของฟังก์ชัน if_else

จากรูปที่ 2.18 การทำงานหากมีการกดสวิตช์ s4 หรือ s4 เป็น 0 จะทำให้เอาต์พุตขา 13 เป็น 1 และเมื่อปล่อยสวิตช์ s4 จะทำให้เอาต์พุตขา 13 เป็น 0

2.8 ตัวแปรในภาษาซี

การเขียนโปรแกรมจำเป็นต้องกำหนดชนิดของตัวแปรให้ถูกต้องสำหรับการใช้งาน เช่น ตัวแปรแบบจำนวนเต็ม ตัวแปรแบบทศนิยม หรือตัวแปรที่สามารถเก็บชนิดของข้อมูลได้ตามต้องการ ซึ่งชนิดของตัวแปรที่ใช้งานในภาษาซีมีดังนี้

ตารางที่ 2.3 ตัวแปรในภาษาซี

ชนิดของตัวแปร	ขนาด	ช่วงข้อมูลที่เก็บได้	การประกาศ
1) Character	8 บิต	–128 ถึง +127	char
2) Unsigned Character	8 บิต	0 ถึง 255	unsigned char
3) Integer	16 บิต	– 32768 ถึง +32767	int
4) Unsigned Integer	16 บิต	0 ถึง 65535	unsigned int
5) Long Integer	32 บิต	–2147483648 ถึง +2147483647	long int
6) Floating Point	32 บิต	1.2×10^{-38} ถึง $3.4 \times 10^{+38}$	float
7) Double	64 บิต	2.3×10^{-308} ถึง $1.7 \times 10^{+308}$	double

1) Character ตัวแปรขนาด 8 บิต ใช้เก็บข้อมูลชนิดตัวอักษรหรือจำนวนเต็มที่มีค่าอยู่ในช่วง –128 ถึง +127

2) Unsigned Character ตัวแปรขนาด 8 บิตแบบไม่คิดเครื่องหมาย ใช้เก็บข้อมูลชนิดตัวอักษรหรือจำนวนเต็มบวกที่มีค่าอยู่ในช่วง 0 ถึง 255

3) Integer ตัวแปรขนาด 16 บิต ใช้เก็บข้อมูลชนิดจำนวนเต็มที่มีค่าอยู่ในช่วง –32768 ถึง +32767

4) Unsigned Integer ตัวแปรขนาด 16 บิต ใช้เก็บข้อมูลชนิดจำนวนเต็มบวกที่มีค่าอยู่ในช่วง 0 ถึง 65535

5) Long Integer ตัวแปรขนาด 32 บิต ใช้เก็บข้อมูลชนิดจำนวนเต็มที่มีค่า –2147483648 ถึง +2147483647

6) Floating Point ตัวแปรขนาด 32 บิต ใช้เก็บข้อมูลเลขทศนิยมในช่วง 1.2×10^{-38} ถึง $3.4 \times 10^{+38}$

7) Double ตัวแปรขนาด 64 บิต ใช้เก็บข้อมูลเลขทศนิยมในช่วง 2.3×10^{-308} ถึง $1.7 \times 10^{+308}$

ตัวอย่างการประกาศตัวแปร

char s1; หมายถึง ประกาศตัวแปร s1 เป็นตัวแปรชนิดตัวอักษรหรือจำนวนเต็มขนาด 8 บิต

float f1,f2; หมายถึง ประกาศตัวแปร f1 และ f2 เป็นตัวแปรชนิดเลขทศนิยม ใช้สำหรับคำนวณเลขที่มีจุดทศนิยม

long int l1,num; หมายถึง ประกาศตัวแปร l1 และ num เป็นตัวแปรชนิดเลขจำนวนเต็มขนาด 32 บิต ใช้สำหรับคำนวณตัวเลขที่มีค่ามาก ๆ

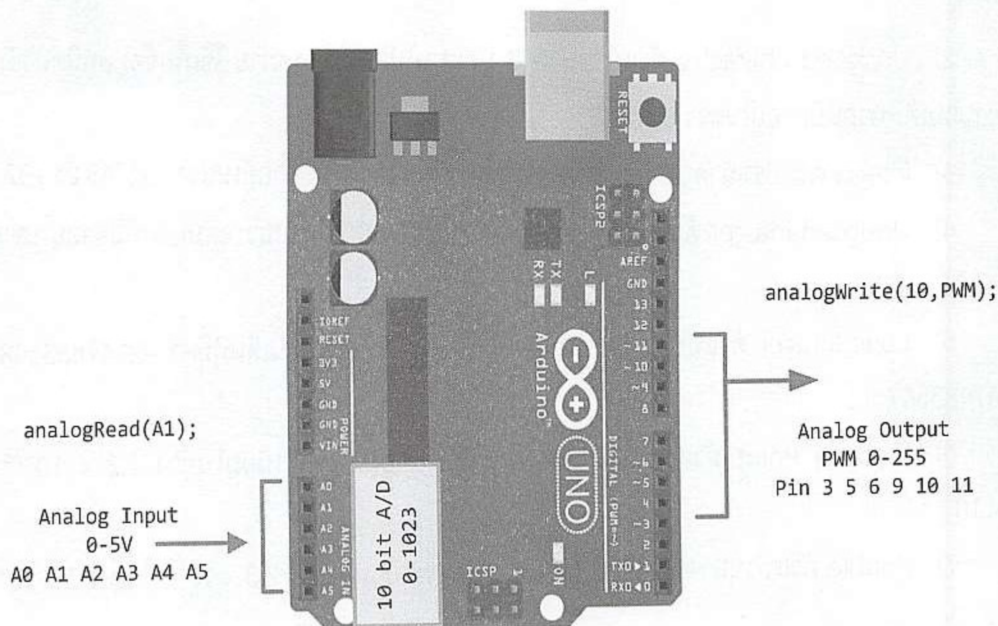
การกำหนดชนิดตัวแปรควรกำหนดให้สอดคล้องกับการใช้งาน เช่น ถ้ามีการคำนวณที่เป็นจุดทศนิยมต้องกำหนดชนิดของตัวแปรเป็นแบบ float

2.9 สรุป

แอนะล็อก คือสัญญาณที่มีความต่อเนื่อง มีหลายค่าแปรเปลี่ยนได้ตามเวลา และเป็นสัญญาณธรรมชาติต่างจากสัญญาณดิจิทัลที่มีเพียง 2 ระดับ สัญญาณแอนะล็อกมีลักษณะเป็นคลื่น มีความถี่และขนาดหรือความเข้มของสัญญาณที่ต่างกันและมักเปลี่ยนแปลงตลอดเวลา เช่น สัญญาณเสียง แสง คลื่น ความชื้น และอุณหภูมิ

analogRead(); คือคำสั่งอ่านค่าข้อมูลแอนะล็อกจากขา A0–A5 ของไมโครคอนโทรลเลอร์ Arduino UNO R3 ซึ่งมีค่าอยู่ระหว่าง 0–1023 โดยผ่านวงจรแปลงสัญญาณแอนะล็อกเป็นดิจิทัลขนาด 10 บิต

analogWrite(); คือคำสั่งส่งข้อมูลแอนะล็อกออกขาเอาต์พุต ซึ่งเป็นการส่งแบบ PWM ค่าที่ส่งออกอยู่ระหว่าง 0–255 ในบอร์ด Arduino UNO จะมีขาเอาต์พุต PWM 6 ขา คือ 3 5 6 9 10 และ 11 โดยสัญญาณ PWM คือสัญญาณที่มีการปรับความกว้างของสัญญาณพัลส์ตามสัดส่วนที่กำหนด



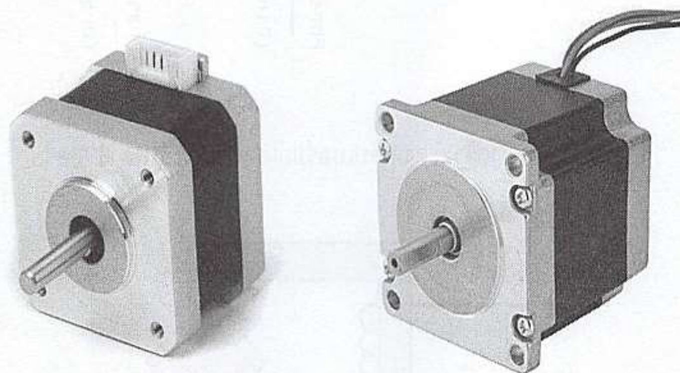
รูปที่ 2.19 แอนะล็อกอินพุตและแอนะล็อกเอาต์พุต

แบบฝึกหัดบทที่ 2

1. สัญญาณแอนะล็อกคืออะไร
2. สัญญาณแอนะล็อกต่างจากสัญญาณดิจิทัลอย่างไร
3. อธิบายฟังก์ชัน analogRead();
4. อธิบายฟังก์ชัน analogWrite();
5. อธิบายฟังก์ชัน map();
6. อธิบายฟังก์ชัน if()
7. ให้เขียนโปรแกรมอ่านค่าความต้านทานแบบ VR ทุก 1 วินาที และความต้านทานแบบ LDR ทุก 2 วินาที และอ่านค่าความชื้นทุก 3 วินาที
8. ให้เขียนโปรแกรมอ่านค่าความต้านทานแบบ VR แล้วแบ่งเป็น 4 ช่วงระดับดังนี้
   - 0–255 ให้เอาต์พุตขา 10 เป็น HIGH
   - 256–511 ให้เอาต์พุตขา 11 เป็น HIGH
   - 512–767 ให้เอาต์พุตขา 12 เป็น HIGH
   - 768–1023 ให้เอาต์พุตขา 13 เป็น HIGH

บทที่ 3 สเต็ปเปอร์มอเตอร์

สเต็ปเปอร์มอเตอร์เป็นมอเตอร์ที่หมุนตามสัญญาณพัลส์ สามารถควบคุมทิศทาง ตำแหน่ง หรือมุมในการหมุนได้ โดยปกติจะหมุนขั้นละ 1.5 1.8 หรือ 2 องศา ซึ่งสามารถดูรายละเอียดได้จากแผ่นป้ายที่ติดกับสเต็ปเปอร์มอเตอร์ การควบคุมสามารถทำได้โดยส่งสัญญาณพัลส์ให้สเต็ปเปอร์มอเตอร์ซึ่งจะทำงานแบบลำดับ (Sequential) ตามสัญญาณพัลส์ สเต็ปเปอร์มอเตอร์มี 2 ชนิด คือ ยูนิโพลาร์ (Unipolar) และไบโพลาร์ (Bipolar)



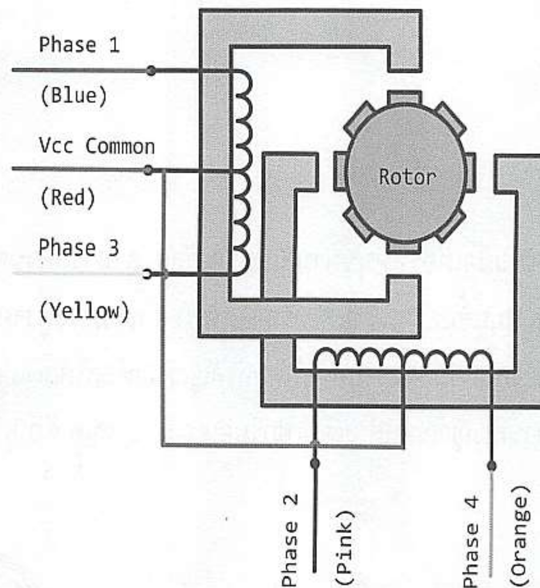
รูปที่ 3.1 สเต็ปเปอร์มอเตอร์

คุณลักษณะของสเต็ปเปอร์มอเตอร์

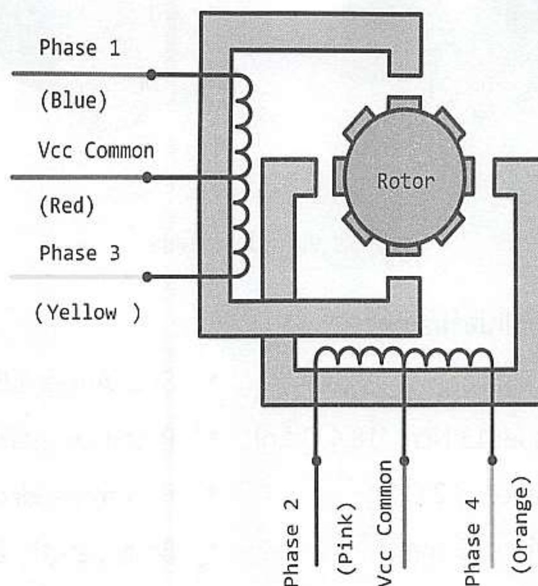
- Motor Type: Bipolar
- Holding Torque: 13 Ncm (18.4 Oz.in)
- Phase Resistance: 3.2 Ohm
- Frame Size: 42 x 42 mm
- Shaft Diameter: 5 mm
- Number of Leads: 4 Lead
- Weight : 180 g
- Step Angle: 1.8°
- Rated Current/Phase: 0.7 A
- Recommended Voltage: 12–24 V
- Body Length: 25 mm
- Shaft Length: 20 mm
- Length: 300 mm

3.1 สเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์

สเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์จะมีสายสัญญาณ 5 สาย และ 6 สาย กรณี 5 สายจะต่อขาร่วม (Common) เข้าด้วยกันและต้องป้อนสัญญาณไฟให้ขาร่วม ส่วนขาสัญญาณควบคุมมี 4 สาย หรือ 4 เฟส การควบคุมต้องป้อนสัญญาณพัลส์แบบลำดับให้กับขาสัญญาณควบคุมเฟส 1 เฟส 2 เฟส 3 และเฟส 4 ซึ่งมีโครงสร้างแสดงดังรูปที่ 3.2 และ 3.3



รูปที่ 3.2 โครงสร้างของสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์ 5 สาย

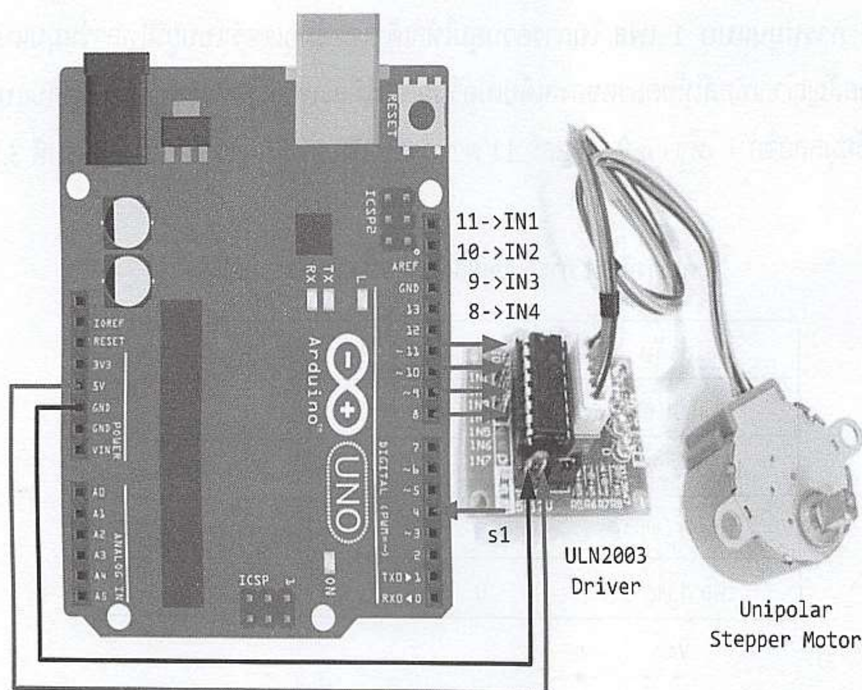


รูปที่ 3.3 โครงสร้างของสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์ 6 สาย

โครงสร้างของสเต็ปเปอร์มอเตอร์มีองค์ประกอบสำคัญ 2 ส่วนใหญ่ ๆ คือ โรเตอร์ (Rotor) ซึ่งเป็นส่วนที่หมุนได้ ทำจากแม่เหล็กถาวร มีลักษณะเป็นทรงกระบอกเซาะร่อง และอีกส่วนคือสเตเตอร์ (Stator) เป็นส่วนที่ติดกับตัวถัง มีขดลวดต่ออนุกรมกัน มีทั้งหมด 4 เฟสซึ่งต้องป้อนสัญญาณพัลส์เข้าเฟส 1 ถึงเฟส 4 สำหรับขาร่วม (Common) อีก 1 หรือ 2 สายต้องต่อกับสัญญาณไฟ

3.2 การเชื่อมต่อ Arduino กับสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์

การเชื่อมต่อ Arduino กับสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์ สัญญาณควบคุมจะถูกส่งออกจากขา 8 9 10 และ 11 เข้าขาอินพุตของไอซี ULN2003 ทำหน้าที่ขยายแรงดันและกระแสไฟฟ้าแล้วส่งสัญญาณไฟออกทางเอาต์พุตเพื่อไปขับสเต็ปเปอร์มอเตอร์ จากตัวอย่างในรูปที่ 3.4 สเต็ปเปอร์มอเตอร์ที่ใช้คือรุ่น 28BYJ-48 ใช้สัญญาณไฟ 5 โวลต์ มุมต่อการหมุนแต่ละครั้งคือ 5.625/64 องศา หรือประมาณ 0.088 องศา



รูปที่ 3.4 วงจรเชื่อมต่อสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์

คุณลักษณะของสเต็ปเปอร์มอเตอร์ 28BYJ-48

- มุมการหมุน 5.625 องศา/64 สเต็ป ให้แรงบิด 60 Nm
- อัตราส่วนการทดเฟือง 1:64 ความต้านทานของขดลวด/เฟส 21 โอห์ม ±10%
- เส้นผ่านศูนย์กลาง 28 มิลลิเมตร ใช้แรงดันไฟฟ้า 5 โวลต์

ตัวอย่างการคำนวณหาจำนวนสัญญาณพัลส์ในการหมุนสเต็ปเปอร์มอเตอร์ 1 รอบหรือ 360 องศา

จำนวนสัญญาณพัลส์ = 360/(5.625/64)
= 360/0.088 = 4096 พัลส์

นั่นคือ ต้องจ่ายสัญญาณพัลส์จำนวน 4096 สเต็ปหรือครั้ง จึงจะทำให้สเต็ปเปอร์มอเตอร์หมุนได้ 1 รอบหรือ 360 องศา หรือการหมุน 1 องศาต้องป้อนสัญญาณพัลส์ 11.38 ครั้ง (4096/360)

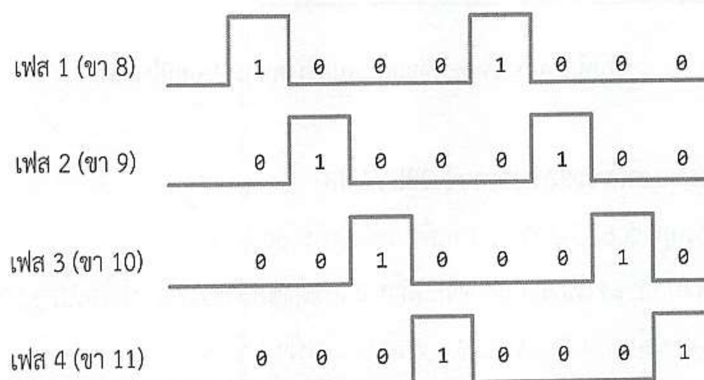
3.3 การควบคุมสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์

การควบคุมสเต็ปเปอร์มอเตอร์ให้หมุนต้องส่งสัญญาณพัลส์หรือสัญญาณไฟฟ้าแบบลำดับให้ขดลวดทั้ง 4 ขดของสเต็ปเปอร์มอเตอร์ ในที่นี้จะกล่าวถึงการควบคุมการหมุน 3 แบบ คือ 1 เฟส 2 เฟส และแบบครึ่งขั้น ดังนี้

1) **การหมุนแบบ 1 เฟส** ในการควบคุมให้สเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์หมุนแบบ 1 เฟส ทำได้โดยจ่ายสัญญาณพัลส์ให้ขดลวดของสเต็ปเปอร์มอเตอร์ทีละเฟสหรือทีละขดลวดเรียงกันแบบต่อเนื่อง ด้วยการส่งข้อมูลลอจิก 1 ให้ขา 8 9 10 และ 11 ตามลำดับ ดังแสดงในตารางที่ 3.1 และรูปที่ 3.5

ตารางที่ 3.1 การควบคุมสเต็ปเปอร์มอเตอร์แบบ 1 เฟส

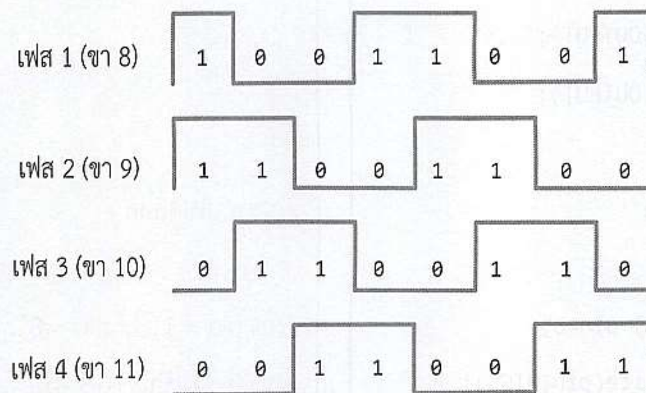
ลำดับ	1	2	3	4
เฟส 1 (ขา 8)	1	0	0	0
เฟส 2 (ขา 9)	0	1	0	0
เฟส 3 (ขา 10)	0	0	1	0
เฟส 4 (ขา 11)	0	0	0	1



รูปที่ 3.5 สัญญาณควบคุมสเต็ปเปอร์มอเตอร์ให้หมุนแบบ 1 เฟส

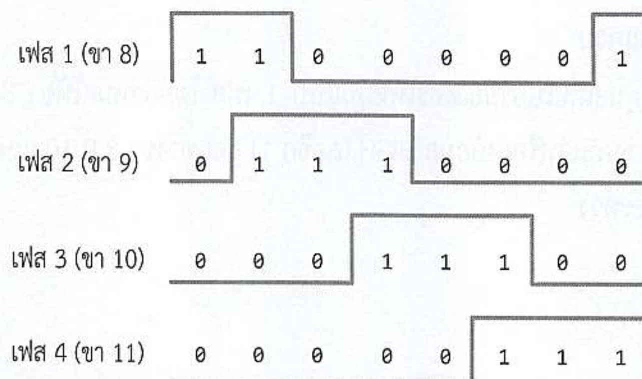
ในการกลับทิศทางหมุนของสเต็ปเปอร์มอเตอร์สามารถทำได้โดยควบคุมการจ่ายสัญญาณพัลส์หรือข้อมูลแบบลำดับจากขา 8 9 10 และ 11 ให้กลับลำดับของขาสัญญาณเป็น 11 10 9 และ 8 ตามลำดับ ก็จะทำให้สเต็ปเปอร์มอเตอร์หมุนกลับทาง

2) การหมุนแบบ 2 เฟส ทำได้โดยการจ่ายสัญญาณพัลส์ให้ขดลวดภายในของสเต็ปเปอร์มอเตอร์ทีละ 2 เฟส หรือทีละ 2 ขดลวดเรียงกันแบบต่อเนื่องตามลำดับ ซึ่งการหมุนแบบ 2 เฟสจะมีแรงบิดมากกว่าการหมุนแบบ 1 เฟส การส่งสัญญาณควบคุมการหมุนแบบ 2 เฟสแสดงดังรูปที่ 3.6



รูปที่ 3.6 สัญญาณควบคุมสเต็ปเปอร์มอเตอร์ให้หมุนแบบ 2 เฟส

3) การหมุนแบบครึ่งขั้น คือการควบคุมแบบผสมกันระหว่างการหมุนแบบ 1 เฟสและ 2 เฟสสลับกันไปมาตามลำดับ 1 ถึง 8 ซึ่งการหมุนแบบครึ่งขั้นจะทำให้มุมหรือองศาในการหมุนละเอียดกว่าถึงสองเท่า ดังแสดงในรูปที่ 3.7



รูปที่ 3.7 สัญญาณควบคุมสเต็ปเปอร์มอเตอร์ให้หมุนแบบครึ่งขั้น

ตัวอย่างที่ 3.1 โปรแกรมควบคุมสเต็ปเปอร์มอเตอร์แบบ 1 เฟส

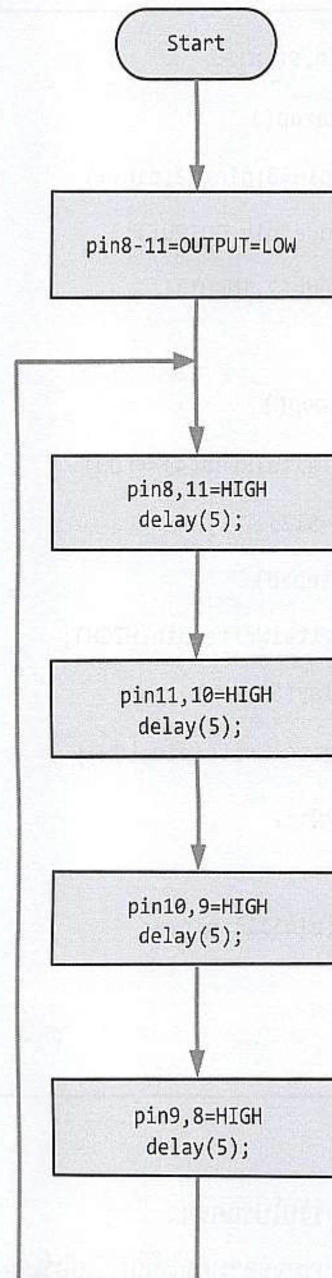
```
int pin;                          //ประกาศตัวแปร pin เป็นชนิดจำนวนเต็ม
void setup()                      //กำหนดค่าและขาสัญญาณ
{
  pinMode(8,OUTPUT);              //กำหนดให้ขา 8-11 เป็นเอาต์พุต
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
}
  void loop()                     //วนรอบไปตลอด
  {
    if(pin>11) pin=8;             //ถ้า pin > 11 ให้ pin = 8
    digitalWrite(pin,HIGH);       //ขา 8-11 เป็น HIGH หรือ 1
    delay(10);                    //หน่วงเวลา 10 ms
    digitalWrite(pin,LOW);        //ขา 8-11 เป็น LOW หรือ 0
    delay(10);                    //หน่วงเวลา 10 ms
    pin++;                        //เลื่อนขาสัญญาณไปตามลำดับ 8-11
  }
```

ผลการรันโปรแกรม

โปรแกรมควบคุมสเต็ปเปอร์มอเตอร์ให้หมุนแบบ 1 เฟส โดยกำหนดให้ขา 8-11 เป็นขาเอาต์พุต จากนั้นวนรอบส่งสัญญาณพัลส์หรือส่งข้อมูล HIGH (ลอจิก 1) ออกทางขา 8 9 10 และ 11 ทีละขาเลื่อนไปเรื่อย ๆ เหมือนไฟวิ่งทีละดวง

ตัวอย่างที่ 3.2 โปรแกรมกลับทิศทางการหมุนสเต็ปเปอร์มอเตอร์แบบ 2 เฟส

```
void setup()
{
  for(int pin=8;pin<12;pin++)
  {
    pinMode(pin,OUTPUT);
    digitalWrite(pin,LOW);
  }
  digitalWrite(8,HIGH);
  digitalWrite(11,HIGH);
}
void loop()
{
  digitalWrite(8,LOW);
  digitalWrite(10,HIGH);
  delay(5);
  digitalWrite(11,LOW);
  digitalWrite(9,HIGH);
  delay(5);
  digitalWrite(10,LOW);
  digitalWrite(8,HIGH);
  delay(5);
  digitalWrite(9,LOW);
  digitalWrite(11,HIGH);
  delay(5);
}
```



รูปที่ 3.8 โฟลว์ชาร์ตกลับทิศทางการหมุนสเต็ปเปอร์มอเตอร์แบบ 2 เฟส

ผลการรันโปรแกรม

โปรแกรมจะควบคุมให้สเต็ปเปอร์มอเตอร์หมุนกลับทิศทางแบบ 2 เฟส โดยการวนรอบส่งข้อมูล HIGH หรือลอจิก 1 ออกทีละคู่ตามลำดับ คือ 1) ขา 8 11 เป็น HIGH 2) ขา 11 10 เป็น HIGH 3) ขา 10 9 เป็น HIGH และ 4) ขา 9 8 เป็น HIGH วนรอบไปเรื่อย ๆ เหมือนไฟวิ่งทีละ 2 ดวง

ตัวอย่างที่ 3.3 โปรแกรมควบคุมสเต็ปเปอร์มอเตอร์ให้หมุนครั้งละ 45 องศาตามกดสวิตช์

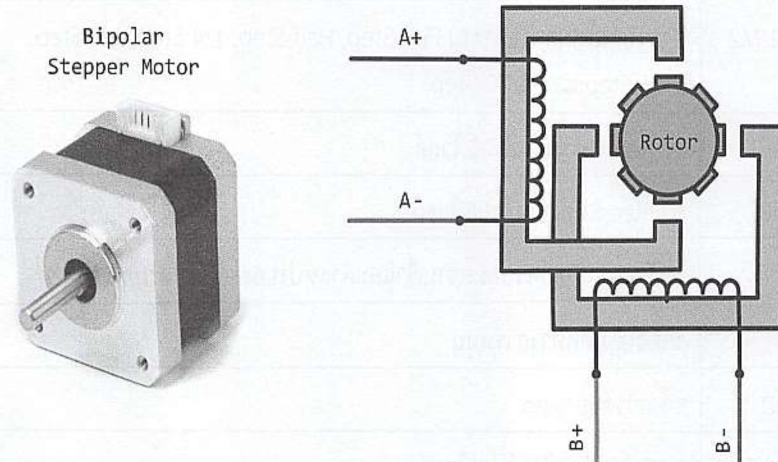
```
int pin,step;
void setup()
{ for(pin=8;pin<12;pin++)          //วนรอบ pin = 8–11
  pinMode(pin,OUTPUT);             //ให้ pin = 8–11 เป็นเอาต์พุต
  pinMode(2,INPUT);                //ให้ขา 2 เป็นอินพุต
}
void loop()
{ if(digitalRead(4)==LOW)          //ถ้าขา 4 ถูกกด
  step=512;                        //ให้ตัวแปร step = 512
  if(step>0)                       //ถ้าตัวแปร step > 0
  { digitalWrite(pin,HIGH);        //ให้ pin 8–11 เป็น HIGH
    delay(5);                      //หน่วงเวลา 5 ms
    digitalWrite(pin,LOW);         //ให้ pin 8–11 เป็น LOW
    step--;                        //ลดค่าตัวแปร step
    pin++;                         //เพิ่มค่าตัวแปร pin
    if(pin>11) pin=8;              //ถ้า pin > 11 ให้ pin = 8
  }
}
```

ผลการรันโปรแกรม

โปรแกรมจะควบคุมให้สเต็ปเปอร์มอเตอร์หมุน 45 องศาตามการกดสวิตช์ที่ขา 4 โดยโปรแกรมจะวนรอบส่งข้อมูล HIGH หรือลอจิก 1 ออกทางขา 8 9 10 และ 11 เป็นจำนวน 512 ครั้ง

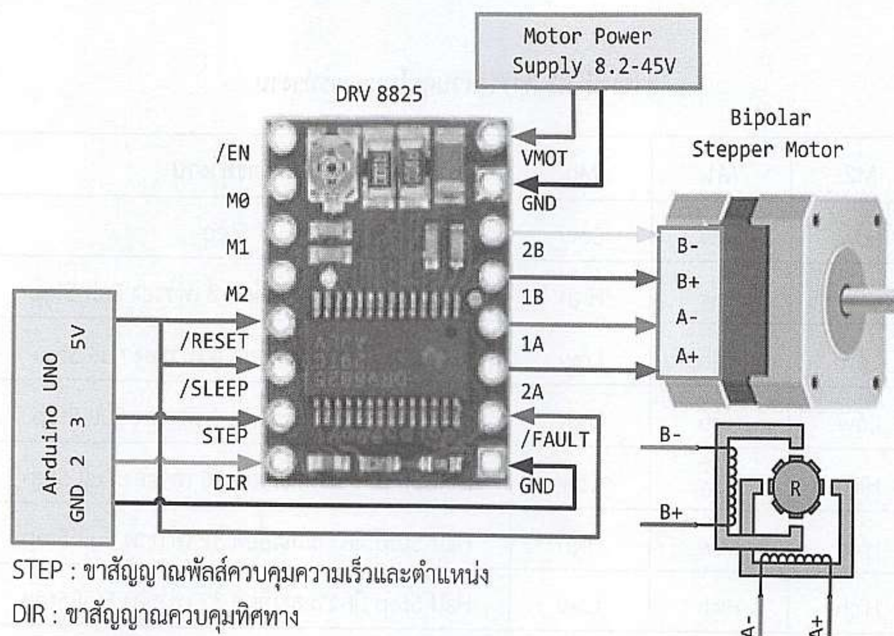
3.4 สเต็ปเปอร์มอเตอร์แบบไบโพลาร์

สเต็ปเปอร์มอเตอร์แบบโพลาร์ เป็นมอเตอร์ที่ขับเคลื่อนด้วยสัญญาณพัลส์แบบลำดับ หมุนรอบแกนได้ 360 องศา มีขดลวด 2 ขด มีสายสัญญาณควบคุม 4 สาย คือ A+ A– B+ และ B–



รูปที่ 3.9 สเต็ปเปอร์มอเตอร์แบบไบโพลาร์และโครงสร้าง

การควบคุมสามารถใช้บอร์ด DRV8825 เพื่อขับเคลื่อนสเต็ปเปอร์มอเตอร์ให้หมุนไปตามตำแหน่งและทิศทางที่ต้องการ แรงดันไฟฟ้าที่ใช้ขับมอเตอร์และบอร์ดมีขนาด 8.2–45 โวลต์ (หากใช้แรงดันไฟฟ้าน้อยกว่านี้วงจรจะไม่ทำงาน) ไฟเลี้ยงวงจรใช้ขนาด 3.3–5 โวลต์ สามารถขับกระแสไฟฟ้าสูงสุดได้ 2.5 แอมแปร์ สามารถปรับความละเอียดของมุมในการหมุนมอเตอร์ได้โดยปรับโหมดการทำงาน M0 M1 และ M2



รูปที่ 3.10 วงจรขับสเต็ปเปอร์มอเตอร์แบบไบโพลาร์

ตารางที่ 3.2 ขาสัญญาณของ DRV8825

ขาสัญญาณ	หน้าที่
/EN	ขาสัญญาณ Enable ต่อกราวด์หรือไม่ต่อก็ได้
M0 M1 M2	ขาควบคุมโหมดการทำงาน Full Step, Half Step, 1/4 Step, 1/8 Step, 1/16 Step และ 1/32 Step
/RESET	ขา Reset ต้องป้อนไฟ 5 โวลต์
/SLEEP	ขา Sleep ต้องป้อนไฟ 5 โวลต์
STEP	ขาสัญญาณพัลส์ควบคุมความเร็วและตำแหน่งของสเต็ปเปอร์มอเตอร์
DIR	ขาควบคุมทิศทางการหมุน
GND	ขากราวด์ของวงจร
/FAULT	ขา Fault ต้องป้อนไฟ 5 โวลต์
2A	ขาขดลวดชุด 2A
1A	ขาขดลวดชุด 1A
2B	ขาขดลวดชุด 2B
1B	ขาขดลวดชุด 1B
GND	ขากราวด์ของแรงดันไฟฟ้าที่จ่ายให้มอเตอร์
VMOT	ขาแรงดันไฟฟ้าต้องป้อนแรงดันให้อยู่ในช่วง 8.2–45 โวลต์

ตารางที่ 3.3 ตารางควบคุมโหมดการทำงาน

M2	M1	M0	โหมดการทำงาน
Low	Low	Low	Full Step
Low	Low	High	Half Step มีความละเอียด 2 เท่าของ Full Step
Low	High	Low	1/4 Step มีความละเอียด 4 เท่าของ Full Step
Low	High	High	1/8 Step มีความละเอียด 8 เท่าของ Full Step
High	Low	Low	Half Step มีความละเอียด 16 เท่าของ Full Step
High	Low	High	Half Step มีความละเอียด 32 เท่าของ Full Step
High	High	Low	Half Step มีความละเอียด 32 เท่าของ Full Step
High	High	High	Half Step มีความละเอียด 32 เท่าของ Full Step

จากตัวอย่างวงจรในรูปที่ 3.10 M2 M1 M0 เป็น LOW ทั้งหมด หมายถึงทำงานที่โหมด Full Step หากใช้มอเตอร์ที่มีความละเอียด 1.8 องศาต่อสเต็ป จะต้องใช้สัญญาณพัลส์ 200 พัลส์ เพื่อให้มอเตอร์หมุน 360 องศาหรือ 1 รอบ (1.8 x 200 = 360 องศา) สามารถเขียนโปรแกรมได้ตามตัวอย่างที่ 3.4

ตัวอย่างที่ 3.4 โปรแกรมควบคุมสเต็ปเปอร์มอเตอร์แบบไบโพลาร์ให้หมุน 1 รอบ

```
int L,DIR=0;                        //ประกาศตัวแปร L และ DIR เท่ากับ 0
void setup()
{
  pinMode(2,OUTPUT); //DIR          //ให้ขา 2 เป็นเอาต์พุต
  pinMode(3,OUTPUT); //STEP         //ให้ขา 3 เป็นเอาต์พุต
}
void loop()
{
  digitalWrite(2,DIR);              //ให้ขา 2 = DIR ควบคุมทิศทาง
  for(L=0;L<200;L++)                //วนรอบ 200 รอบ = 360 องศา
  {
    digitalWrite(3,1);              //ให้ขา 3 เป็นลอจิก 1
    delayMicroseconds(1000);        //หน่วงเวลา 1000 ไมโครวินาที
    digitalWrite(3,0);              //ให้ขา 3 เป็นลอจิก 0
    delayMicroseconds(1000);        //หน่วงเวลา 1000 ไมโครวินาที
  }
  delay(5000);                      //หน่วงเวลา 5000 มิลลิวินาที
}
```

ผลการรันโปรแกรม

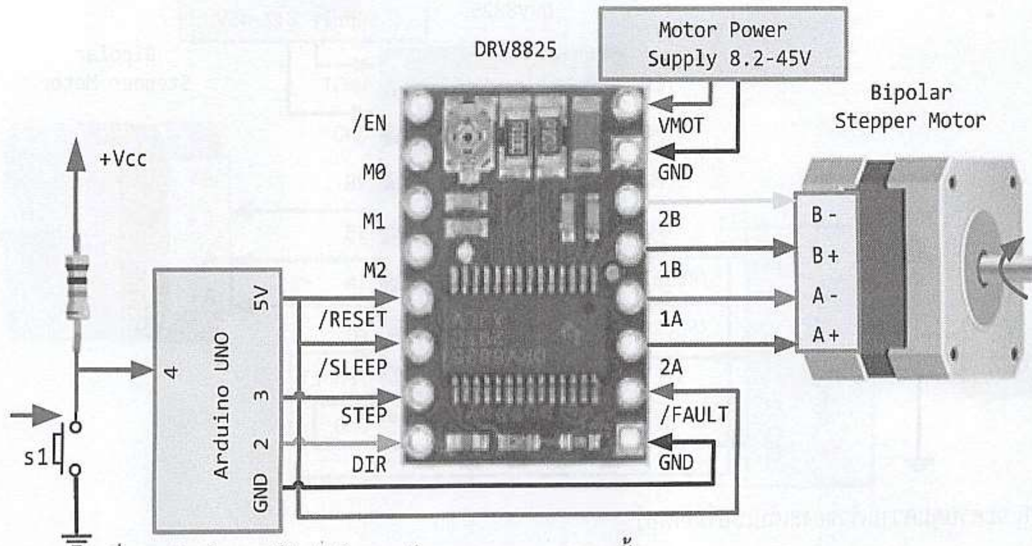
โปรแกรมจะกำหนดให้ขา 2 เป็นลอจิก 0 ซึ่งเป็นขาที่กำหนดทิศทางการหมุนของสเต็ปเปอร์มอเตอร์ และจะวนรอบส่งสัญญาณพัลส์ออกทางขา 3 จำนวน 200 ครั้ง ทำให้มอเตอร์หมุน 360 องศา แล้วหยุด 5 วินาที

ตัวอย่างที่ 3.5 โปรแกรมควบคุมสเต็ปเปอร์มอเตอร์แบบไบโพลาร์ให้หมุน 1 รอบตามการกดสวิตช์

```
int L,DIR=0;                    //ประกาศตัวแปร L และ DIR เท่ากับ 0
void setup()
{ pinMode(2,OUTPUT); //DIR      //ให้ขา 2 เป็นเอาต์พุต
  pinMode(3,OUTPUT); //STEP     //ให้ขา 3 เป็นเอาต์พุต
  pinMode(4,INPUT);             //ให้ขา 4 เป็นอินพุต
}
void loop()
{ digitalWrite(2,DIR);          //ให้ขา 2 = DIR ควบคุมทิศทาง
  int s1=digitalRead(4);        //อ่านค่าการกดสวิตช์จากขา 4
  if(s1==0) L=200;              //ถ้ากดสวิตช์ s1 ให้ L = 200 หมุน 1 รอบ
  if(L>200)                     //ถ้า L มากกว่า 200
  { digitalWrite(3,1);          //ให้ขา 3 เป็นลอจิก 1
    delayMicroseconds(1000);    //หน่วงเวลา 1000 ไมโครวินาที
    digitalWrite(3,0);          //ให้ขา 3 เป็นลอจิก 0
    delayMicroseconds(1000);    //หน่วงเวลา 1000 ไมโครวินาที
    L=L-1;                      //ลดค่า L
  }
  delay(5000);                  //หน่วงเวลา 5000 มิลลิวินาที
}
```

ผลการรันโปรแกรม

โปรแกรมจะกำหนดให้ขา 2 (DIR) เป็นลอจิก 0 ซึ่งเป็นขาที่กำหนดทิศทางการหมุนของสเต็ปเปอร์มอเตอร์ เมื่อมีการกดสวิตช์ s1 ที่ต่อกับขา 4 ตัวแปร L = 200 โปรแกรมจะวนรอบส่งสัญญาณพัลส์ออกขา 3 จำนวน 200 ครั้ง ทำให้มอเตอร์หมุน 360 องศาแล้วหยุด และจะเริ่มทำงานใหม่เมื่อมีการกดสวิตช์



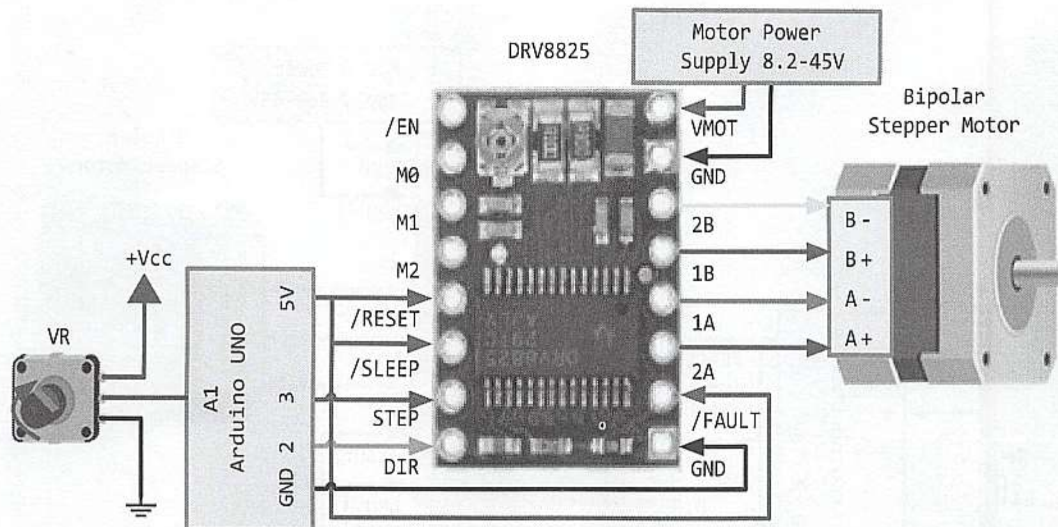
รูปที่ 3.11 วงจรสวิตช์ควบคุมการหมุนของสเต็ปเปอร์มอเตอร์แบบไบโพลาร์

ตัวอย่างที่ 3.6 โปรแกรมควบคุมความเร็วของสเต็ปเปอร์มอเตอร์ตามการปรับค่า VR

`int L,DIR=1;`	//ประกาศตัวแปร L และ DIR เท่ากับ 1
`void setup()`	
`{ pinMode(2,OUTPUT); //DIR`	//ให้ขา 2 เป็นเอาต์พุต
`  pinMode(3,OUTPUT); //STEP`	//ให้ขา 3 เป็นเอาต์พุต
`}`	
`void loop()`	
`{ digitalWrite(2,DIR);`	//ให้ขา 2 = DIR ควบคุมทิศทาง
`  int VR=analogRead(A1);`	//อ่านค่า 0–1023 จาก VR
`  digitalWrite(3,1);`	//ให้ขา 3 เป็นลอจิก 1
`  delayMicroseconds(200+VR);`	//หน่วงเวลา 200 + VR ควบคุมความเร็ว
`  digitalWrite(3,0);`	//ให้ขา 3 เป็นลอจิก 0
`  delayMicroseconds(200+VR);`	//หน่วงเวลา 200 + VR ควบคุมความเร็ว
`}`	

ผลการรันโปรแกรม

กำหนดให้ขา 2 (*DIR*) เป็นลอจิก 1 ซึ่งเป็นขาที่กำหนดทิศทางการหมุนของสเต็ปเปอร์มอเตอร์ โปรแกรมจะวนรอบส่งสัญญาณพัลส์ออกขา 3 ทำให้มอเตอร์หมุน และสามารถปรับความเร็วในการหมุนด้วยคำสั่ง delayMicroseconds (200+VR); ตามการปรับค่าของ VR จากขาแอนะล็อก A1

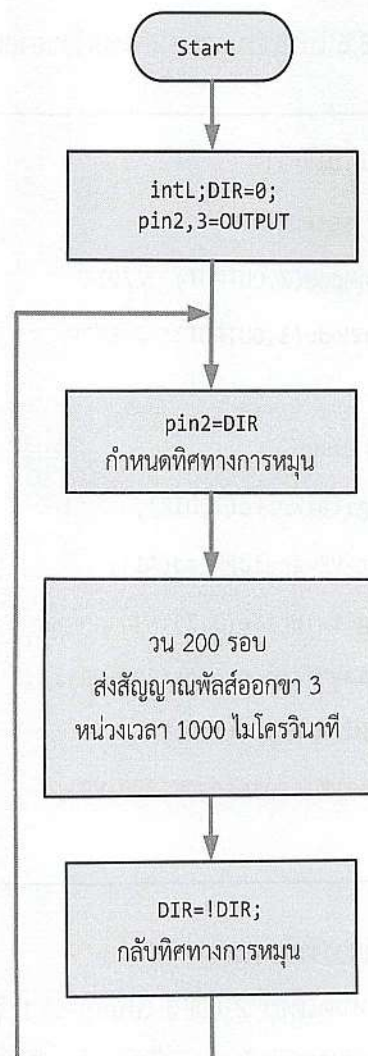


VR จะควบคุมความเร็วของสเต็ปเปอร์มอเตอร์

รูปที่ 3.12 วงจรควบคุมความเร็วของสเต็ปเปอร์มอเตอร์ตามการปรับค่า VR

ตัวอย่างที่ 3.7 โปรแกรมควบคุมสเต็ปเปอร์มอเตอร์แบบไบโพลาร์ให้หมุนไปกลับ

```
int L,DIR=0;
void setup()
  {
    pinMode(2,OUTPUT); //DIR
    pinMode(3,OUTPUT); //STEP
  }
void loop()
{
  digitalWrite(2,DIR);
  for(L=0;L<200;L++)
  {
    digitalWrite(3,1);
    delayMicroseconds(1000);
    digitalWrite(3,0);
    delayMicroseconds(1000);
  }
  DIR=!DIR;
  delay(5000);
}
```



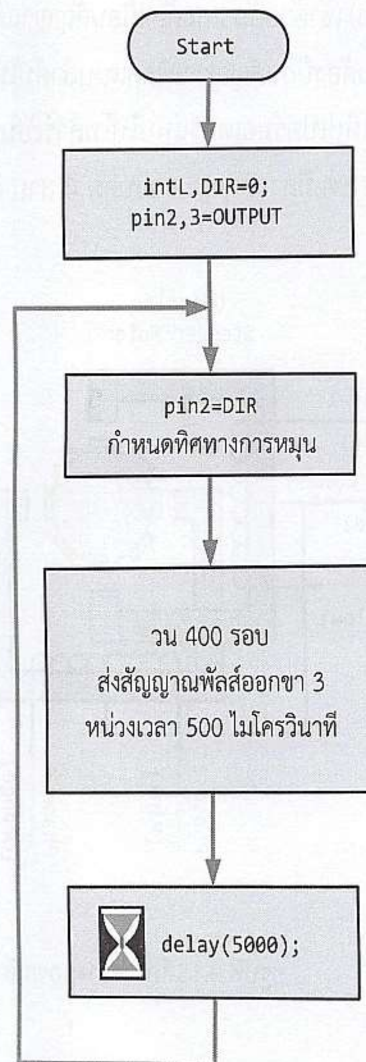
รูปที่ 3.13 โฟลว์ชาร์ตกลับทิศทางการหมุนสเต็ปเปอร์มอเตอร์

ผลการรันโปรแกรม

โปรแกรมจะวนรอบส่งสัญญาณพัลส์ออกขา 3 จำนวน 200 ครั้ง ทำให้มอเตอร์หมุน 360 องศา แล้วทำการกลับทิศทางการหมุนของสเต็ปเปอร์มอเตอร์โดยกลับค่าตัวแปร DIR ให้เป็น 0 หรือ 1

ตัวอย่างที่ 3.8 โปรแกรมควบคุมสเต็ปเปอร์มอเตอร์แบบไบโพลาร์ให้หมุน 1 รอบ Mode 400 พัลส์ต่อรอบ

```
int L,DIR=0;
void setup()
{ pinMode(2,OUTPUT); //DIR
  pinMode(3,OUTPUT); //STEP
}
void loop()
{
  digitalWrite(2,DIR);
  for(L=0;L<400;L++)
  {
    digitalWrite(3,1);
    delayMicroseconds(500);
    digitalWrite(3,0);
    delayMicroseconds(500);
  }
  delay(5000);
}
```



รูปที่ 3.14 โฟลว์ชาร์ตการควบคุมสเต็ปเปอร์มอเตอร์ Mode 400

ผลการรันโปรแกรม

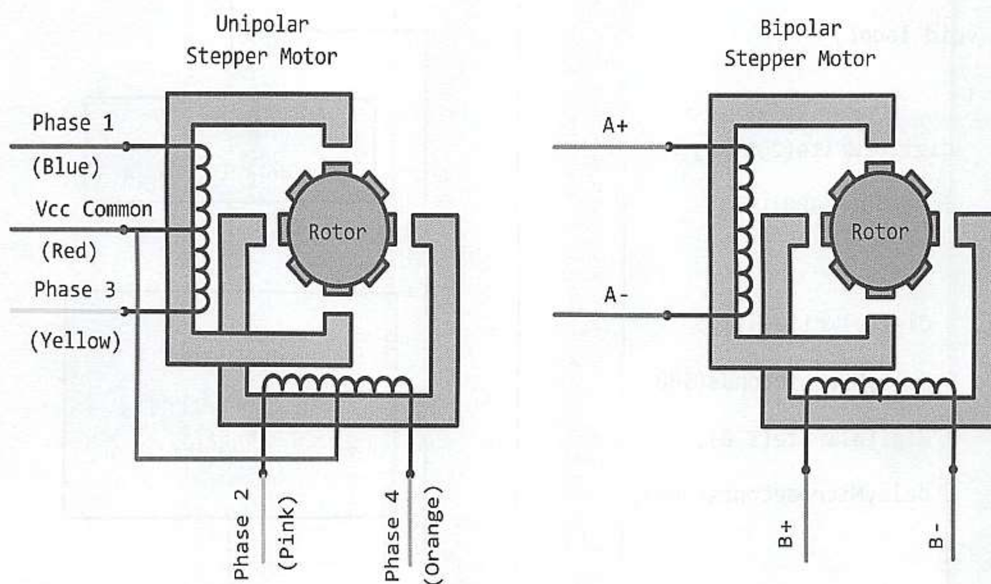
โปรแกรมจะกำหนดให้ขา 2 เป็นลอจิก 0 ซึ่งเป็นขาที่กำหนดทิศทางการหมุนของสเต็ปเปอร์มอเตอร์ และวนรอบส่งสัญญาณพัลส์ออกขา 3 จำนวน 400 ครั้ง ทำให้มอเตอร์หมุน 360 องศา ซึ่งจะควบคุมสเต็ปเปอร์มอเตอร์แบบไบโพลาร์ Mode 400 พัลส์ต่อรอบ (M2 = Low, M1 = Low, M0 = High)

3.5 สรุป

สเต็ปเปอร์มอเตอร์เป็นมอเตอร์ที่หมุนตามสัญญาณพัลส์ สามารถควบคุมทิศทาง ตำแหน่ง หรือมุมในการหมุนได้ โดยปกติจะหมุนขั้นละ 1.5 1.8 หรือ 2 องศา การควบคุมสามารถทำได้โดยส่งสัญญาณพัลส์ให้สเต็ปเปอร์มอเตอร์ซึ่งจะทำงานแบบลำดับตามสัญญาณพัลส์ โดยสเต็ปเปอร์มอเตอร์มี 2 ชนิด คือ ยูนิโพลาร์และไบโพลาร์

สเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์มีสายสัญญาณ 5 สาย และ 6 สาย กรณี 5 สายจะต่อขาร่วม (Common) เข้าด้วยกัน และต้องป้อนสัญญาณไฟให้กับขาร่วม ส่วนขาสัญญาณควบคุมมี 4 สาย หรือ 4 เฟส การควบคุมต้องป้อนสัญญาณพัลส์แบบลำดับให้ขาสัญญาณควบคุมเฟส 1 เฟส 2 เฟส 3 และเฟส 4

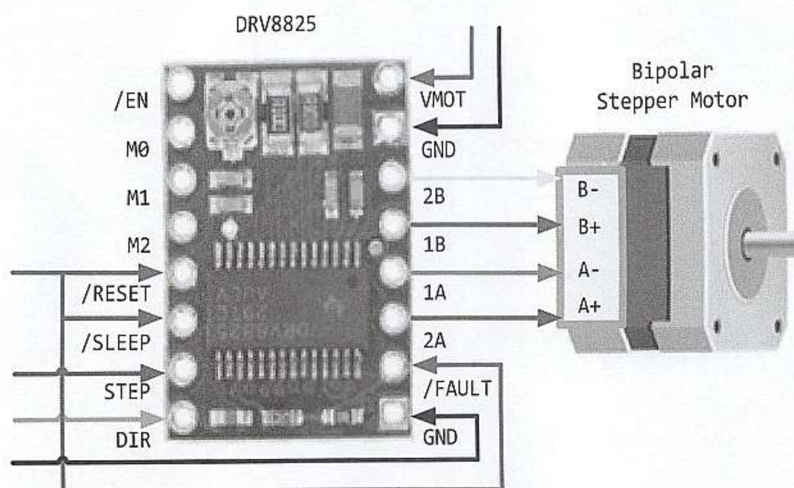
สเต็ปเปอร์มอเตอร์แบบไบโพลาร์เป็นมอเตอร์ที่ขับเคลื่อนด้วยพัลส์ หมุนรอบแกนได้ 360 องศา มีขดลวด 2 ขด มีสายสัญญาณควบคุม 4 สาย คือ A+ A– B+ และ B– มีโครงสร้างแสดงดังรูป



รูปที่ 3.15 โครงสร้างของสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์และไบโพลาร์

แบบฝึกหัดบทที่ 3

1. ให้เขียนโครงสร้างของสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์
2. ให้เขียนโครงสร้างของสเต็ปเปอร์มอเตอร์แบบไบโพลาร์
3. ให้เขียนโปรแกรมควบคุมสเต็ปเปอร์มอเตอร์แบบยูนิโพลาร์แบบครึ่งขั้น
4. ให้คำนวณหาจำนวนรอบในการส่งสัญญาณพัลส์เพื่อให้มอเตอร์หมุน 2 รอบ
5. อธิบายขาสัญญาณของ DRV8825



รูปที่ 3.16 วงจรขับสเต็ปเปอร์มอเตอร์แบบไบโพลาร์

6. อธิบายโหมดการทำงานของ DRV8825
7. ให้เขียนโปรแกรมควบคุมการหมุนของสเต็ปเปอร์มอเตอร์ให้หมุนครั้งละ 180 องศาตามการกดสวิตช์

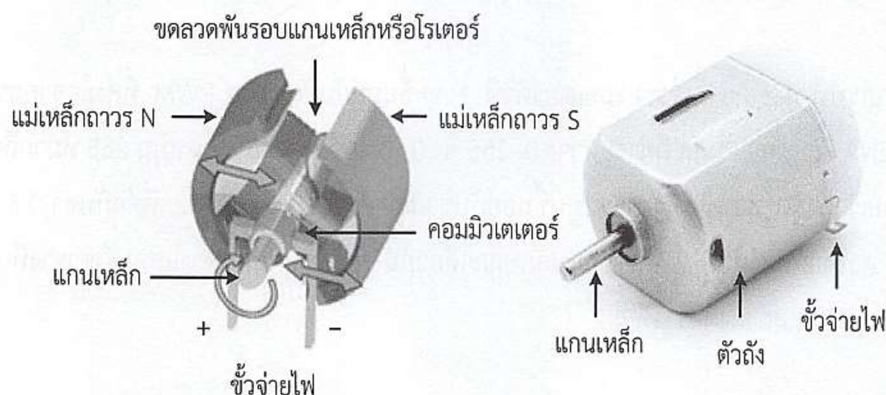
บทที่ 4 ดีซีมอเตอร์และเอนโค้ดเดอร์มอเตอร์

มอเตอร์ คือเครื่องกลไฟฟ้าที่ทำหน้าที่เปลี่ยนพลังงานไฟฟ้าให้เป็นพลังงานกล มีทั้งแบบใช้ไฟฟ้ากระแสสลับและไฟฟ้ากระแสตรง การทำงานของมอเตอร์จะใช้แรงของสนามแม่เหล็กทำให้โรเตอร์ (Rotor) หมุนเคลื่อนที่เพื่อขับเคลื่อนกลไกหรือระบบที่ต้องการควบคุม

ในบทนี้จะกล่าวถึงการควบคุมมอเตอร์ไฟฟ้ากระแสตรง วงจรขับมอเตอร์กระแสตรง และเอนโค้ดเดอร์มอเตอร์

4.1 ดีซีมอเตอร์

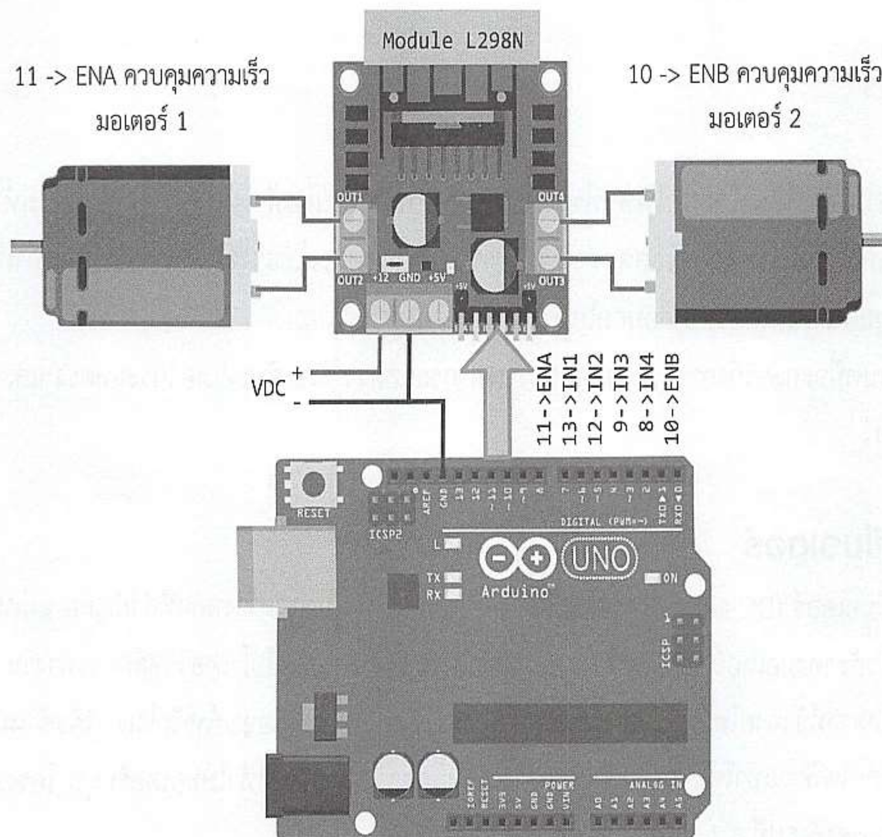
ดีซีมอเตอร์ (DC Motor) หรือมอเตอร์ไฟฟ้ากระแสตรงมีโครงสร้างหลักที่สำคัญคือ แม่เหล็กถาวรซึ่งติดรอบตัวถังของมอเตอร์ และขดลวดทองแดงที่พันรอบแกนเหล็กหรือโรเตอร์ หลักการทำงาน เมื่อจ่ายไฟฟ้ากระแสตรงที่ขั้วจ่ายไฟโดยผ่านคอมมิวเตเตอร์ (Commutator) เพื่อจ่ายให้กับโรเตอร์ซึ่งทำหน้าที่สร้างสนามแม่เหล็กไฟฟ้าจะทำให้เกิดแรงผลักระหว่างแม่เหล็กถาวรกับโรเตอร์ทำให้มอเตอร์หมุน โครงสร้างของดีซีมอเตอร์แสดงดังรูปที่ 4.1



รูปที่ 4.1 ดีซีมอเตอร์และโครงสร้างภายใน

4.2 บอร์ดขับมอเตอร์ L298N

บอร์ด Motor Drive Module L298N สามารถขับดีซีมอเตอร์ได้ 2 ตัว ควบคุมทิศทางการหมุนของมอเตอร์ได้ที่ขาอินพุต IN1–IN4 และควบคุมความเร็วมอเตอร์แบบ PWM ที่ขา ENA และ ENB ใช้แรงดันไฟฟ้าได้ 7–35 โวลต์ จ่ายกระแสไฟฟ้าสูงสุดข้างละ 2 แอมแปร์ มีแหล่งจ่ายไฟ 5 โวลต์ในบอร์ด การควบคุมทิศทางของดีซีมอเตอร์ตัวที่ 1 ขึ้นอยู่กับขา IN1–IN2 ส่วนการควบคุมความเร็วในการหมุนของมอเตอร์จะขึ้นอยู่กับขา A Enable ส่วนมอเตอร์ตัวที่ 2 ก็ทำงานในลักษณะเดียวกัน



รูปที่ 4.2 วงจรควบคุมความเร็วและทิศทางของดีซีมอเตอร์

จากรูปที่ 4.2 ความเร็วของมอเตอร์ตัวที่ 1 จะขึ้นอยู่กับสัญญาณ PWM ที่ส่งมาจากขา 11 ต่อเข้ากับขา ENA สัญญาณ PWM มีค่าระหว่าง 0–255 ซึ่ง 0 หมายถึงมอเตอร์หยุดหมุน 255 หมายถึงมอเตอร์หมุนที่ความเร็วสูงสุด ส่วนทิศทางการหมุนขึ้นอยู่กับขาสัญญาณ IN1 และ IN2 ที่ต่อกับขา 13 และ 12 ตามลำดับ ส่วนมอเตอร์ตัวที่ 2 ก็ทำงานในลักษณะเดียวกัน การควบคุมทิศทางแสดงดังตารางที่ 4.1 และการต่อขาสัญญาณแสดงดังตารางที่ 4.2

ตารางที่ 4.1 การควบคุมทิศทางและความเร็ว

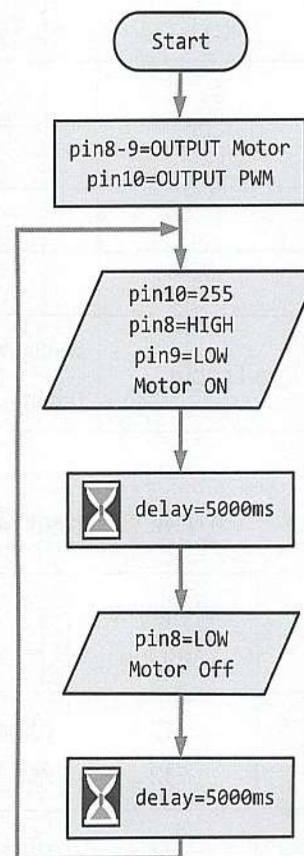
IN1	IN2	มอเตอร์ 1
0	0	หยุดหมุน
0	1	หมุนตามเข็มนาฬิกา
1	0	หมุนทวนเข็มนาฬิกา
1	1	หยุดหมุน
A Enable	ส่งสัญญาณ PWM 0-255 เพื่อควบคุมความเร็วของมอเตอร์ 0 คือแรงดันไฟต่ำสุด ส่วน 255 คือแรงดันไฟสูงสุด	

ตารางที่ 4.2 การต่อขาสัญญาณและการทำงานของบอร์ด L298N

ขาสัญญาณ L298N	ขาสัญญาณ Arduino	หน้าที่
IN1 IN2	13 12	เป็นขาอินพุต ต่อกับบอร์ด Arduino เพื่อควบคุมทิศทางของดีซีมอเตอร์ตัวที่ 1
A Enable	11	เป็นขาอินพุต ต่อกับบอร์ด Arduino เพื่อควบคุมความเร็วแบบ PWM ของมอเตอร์ตัวที่ 1 หากต่อกับสัญญาณไฟ 5 โวลต์จะได้ความเร็วสูงสุดของมอเตอร์
OUT1 OUT2		ขาสัญญาณเอาต์พุต ต่อกับมอเตอร์ตัวที่ 1
B Enable	10	เป็นขาอินพุต ต่อกับบอร์ด Arduino เพื่อควบคุมความเร็วแบบ PWM ของมอเตอร์ตัวที่ 2 หากต่อกับสัญญาณไฟ 5 โวลต์จะได้ความเร็วสูงสุดของมอเตอร์
IN3 IN4	9 8	เป็นขาอินพุต ต่อกับบอร์ด Arduino เพื่อควบคุมทิศทางของดีซีมอเตอร์ตัวที่ 2
OUT3 OUT4		ขาสัญญาณเอาต์พุต ต่อกับมอเตอร์ตัวที่ 2
12V		ขาอินพุตป้อนแรงดันไฟฟ้า 6-12 โวลต์จากแหล่งจ่ายไฟ
GND	GND	ขาอินพุต ต่อกราวด์จากแบตเตอร์รี่และต่อเข้ากับบอร์ด Arduino
5V		ขาเอาต์พุตแรงดันไฟฟ้า 5 โวลต์

ตัวอย่างที่ 4.1 โปรแกรมควบคุมมอเตอร์ให้หมุนและหยุดหมุนสลับกัน

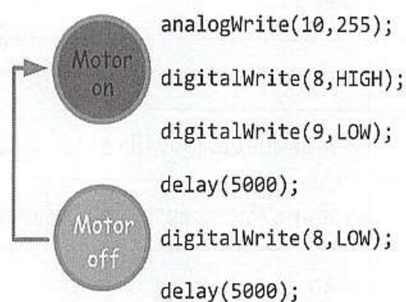
```
void setup()
{ pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
}
void loop()
  {
    analogWrite(10,255);
    digitalWrite(8,HIGH);
    digitalWrite(9,LOW);
    delay(5000);
    digitalWrite(8,LOW);
    delay(5000);
  }
```



รูปที่ 4.3 โฟลว์ชาร์ตการปิด-เปิดมอเตอร์

ผลการรันโปรแกรม

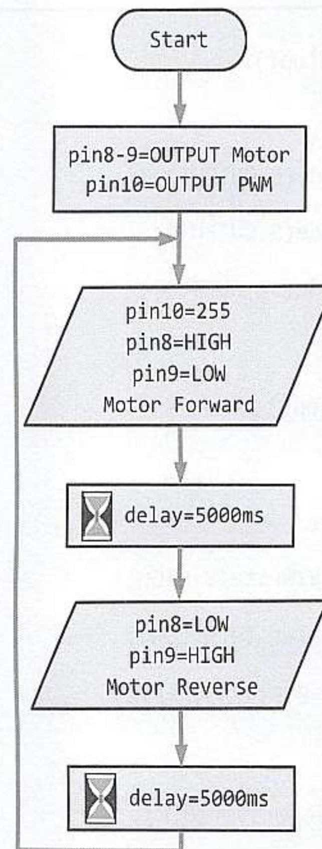
โปรแกรมส่งสัญญาณให้ขา 10 เท่ากับ 255 ให้มอเตอร์หมุนด้วยความเร็วสูงสุด และส่งสัญญาณให้ขา 8 เป็น HIGH หรือลอจิก 1 และขา 9 เป็น LOW หรือลอจิก 0 ทำให้มอเตอร์หมุนเป็นเวลา 5000 มิลลิวินาที หรือ 5 วินาที จากนั้นให้ขา 8 เป็น LOW มอเตอร์หยุดหมุนเป็นเวลา 5 วินาทีสลับกันไปมา



รูปที่ 4.4 มอเตอร์หมุนและหยุดหมุนสลับกันอย่างละ 5 วินาที

ตัวอย่างที่ 4.2 โปรแกรมกลับทิศทางการหมุนของมอเตอร์

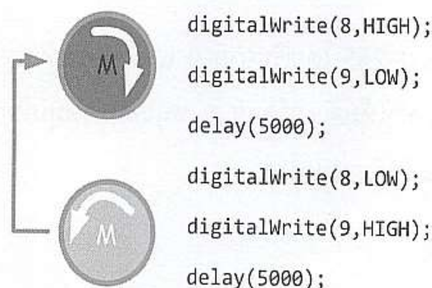
```
void setup()
{ pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);  //PWM
}
void loop()
  { analogWrite(10,255);
    digitalWrite(8,HIGH);
    digitalWrite(9,LOW);
    delay(5000);
    digitalWrite(8,LOW);
    digitalWrite(9,HIGH);
    delay(5000);
  }
```



รูปที่ 4.5 โฟลว์ชาร์ตกลับทิศทางการหมุนมอเตอร์

ผลการรันโปรแกรม

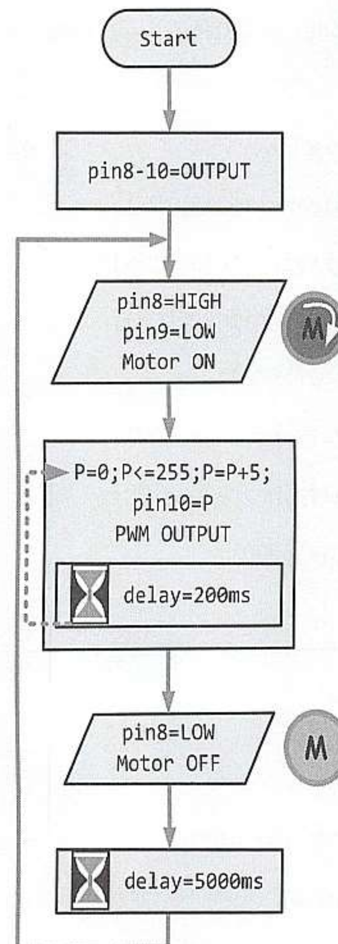
โปรแกรมส่งสัญญาณให้ขา 10 เท่ากับ 255 ให้มอเตอร์หมุนด้วยความเร็วสูงสุด และส่งสัญญาณให้ขา 8 เป็น HIGH หรือลอจิก 1 และขา 9 เป็น LOW หรือลอจิก 0 ทำให้มอเตอร์หมุนเป็นเวลา 5000 มิลลิวินาที หรือ 5 วินาที จากนั้นให้มอเตอร์กลับทิศทางการหมุนโดยขา 8 เป็น LOW ขา 9 เป็น HIGH เป็นเวลา 5 วินาทีสลับกันไปมา



รูปที่ 4.6 มอเตอร์หมุนกลับทิศทางสลับกันอย่างละ 5 วินาที

ตัวอย่างที่ 4.3 โปรแกรมควบคุมความเร็วของมอเตอร์แบบ PWM

```
void setup()
{
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
}
void loop()
{
  int P;
  digitalWrite(8,HIGH);
  digitalWrite(9,LOW);
  for(P=0;P<=255;P=P+5)
  {
    analogWrite(10,P);
    delay(200);
  }
  digitalWrite(8,LOW);
  delay(5000);
}
```



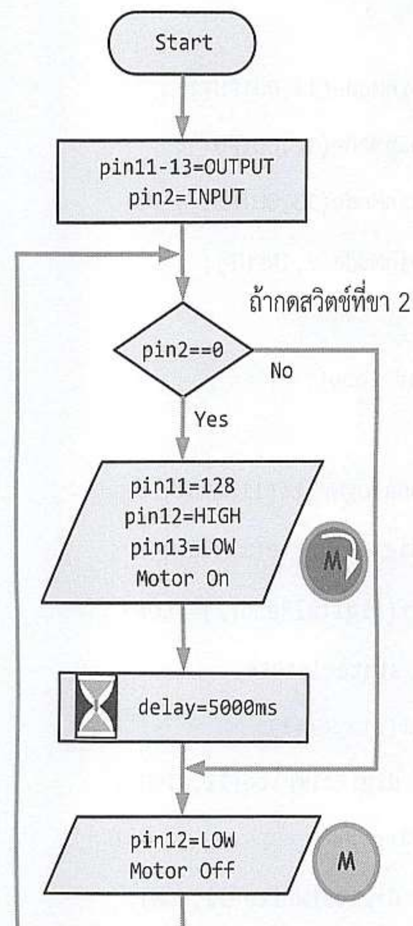
รูปที่ 4.7 โฟลว์ชาร์ตการควบคุมมอเตอร์แบบ PWM

ผลการรันโปรแกรม

โปรแกรมวนรอบส่งข้อมูล 0–255 โดยเริ่มจาก 0 และเพิ่มขึ้นทีละ 5 จนถึง 255 ออกทางขา 10 ซึ่งเป็นการส่งสัญญาณแบบ PWM ทำให้มอเตอร์ค่อย ๆ หมุนจนถึงหมุนที่ความเร็วสูงสุด (PWM = 255) แล้วหยุดทำงานเป็นเวลา 5 วินาทีแล้วเริ่มหมุนใหม่

ตัวอย่างที่ 4.4 โปรแกรมควบคุมให้มอเตอร์ทำงาน 5 วินาทีตามการกดสวิตช์

```
void setup()
{
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
  pinMode(2,INPUT);
}
void loop()
{
  if(digitalRead(2)==LOW)
  {
    analogWrite(11,128);
    digitalWrite(12,HIGH);
    digitalWrite(13,LOW);
    delay(5000);
  }
  digitalWrite(12,LOW);
}
```



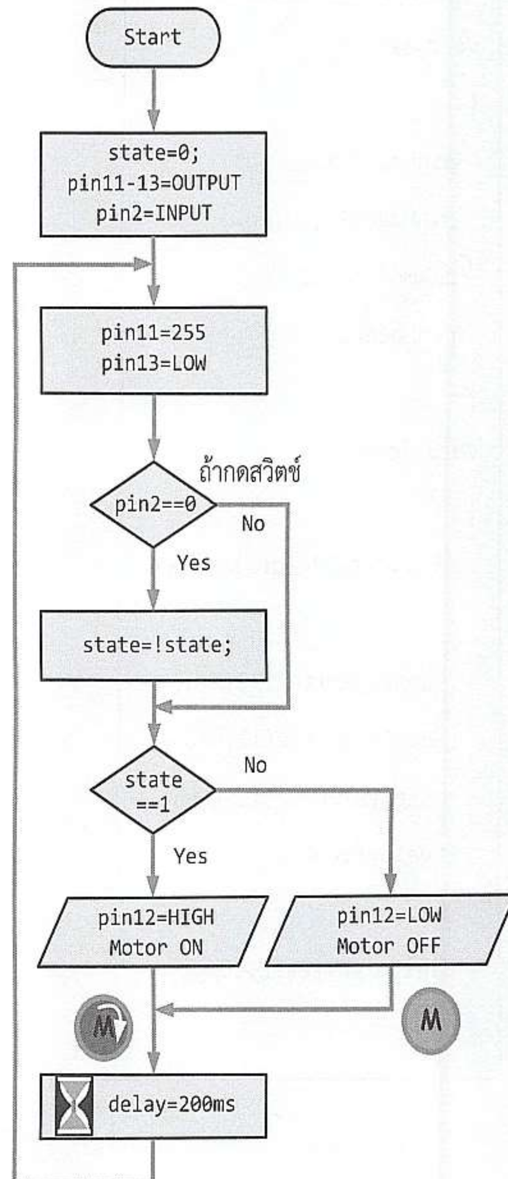
รูปที่ 4.8 โฟลว์ชาร์ตควบคุมให้มอเตอร์ทำงานตามการกดสวิตช์

ผลการรันโปรแกรม

เมื่อกดสวิตช์ที่ขา 2 จะทำให้ขา 12 เป็น HIGH และขา 13 เป็น LOW ทำให้มอเตอร์ 2 (ต่อกับขา 11–13) หมุนด้วยความเร็ว PWM = 128 เป็นเวลา 5 วินาทีแล้วหยุด

ตัวอย่างที่ 4.5 โปรแกรมปิด-เปิดมอเตอร์ตามการกดสวิตช์

```
int state=0;
void setup()
{
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
  pinMode(2,INPUT);
}
void loop()
{
  analogWrite(11,255);
  digitalWrite(13,LOW)
  if(digitalRead(2)==LOW)
    state=!state;
  if(state==1)
    digitalWrite(12,HIGH);
  else
    digitalWrite(12,LOW);
  delay(200);
}
```



รูปที่ 4.9 โฟลว์ชาร์ตการปิด-เปิดมอเตอร์ตามการกดสวิตช์

ผลการรันโปรแกรม

ทุกครั้งที่กดสวิตช์ที่ขา 2 จะกลับสถานะของตัวแปร state ถ้าตัวแปร state เท่ากับ 1 ให้มอเตอร์หมุน แต่ถ้าตัวแปร state เท่ากับ 0 ให้มอเตอร์หยุดหมุน

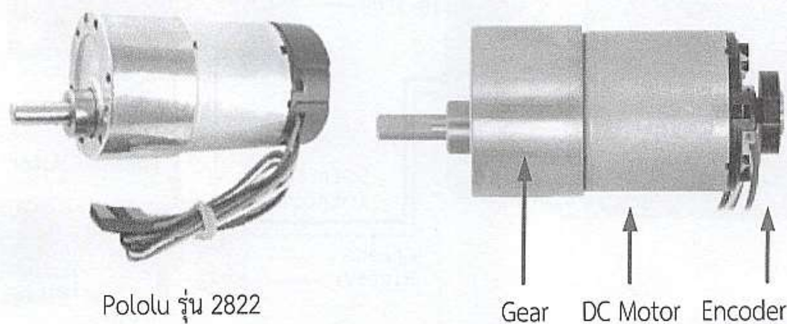
4.3 เอนโค้ดเดอร์มอเตอร์

เอนโค้ดเดอร์มอเตอร์ (Encoder Motor) คือดีซีมอเตอร์ที่มีเฟืองทดรอบเพื่อเพิ่มแรงบิด และมีวงจรเอนโค้ดเดอร์ที่ใช้ตรวจวัดจำนวนรอบการหมุนเพื่อควบคุมหรือนับจำนวนรอบในการหมุนของมอเตอร์ เอนโค้ดเดอร์จะมี 3 ส่วนหลักคือ

ส่วนที่ 1 ชุดเฟือง (Gear) ทำหน้าที่ทดรอบเพิ่มแรงบิดให้มอเตอร์

ส่วนที่ 2 ดีซีมอเตอร์ซึ่งเป็นมอเตอร์ไฟฟ้ากระแสตรง

ส่วนที่ 3 เอนโค้ดเดอร์ เป็นวงจรนับสัญญาณพัลส์หรือสัญญาณเอนโค้ดเดอร์

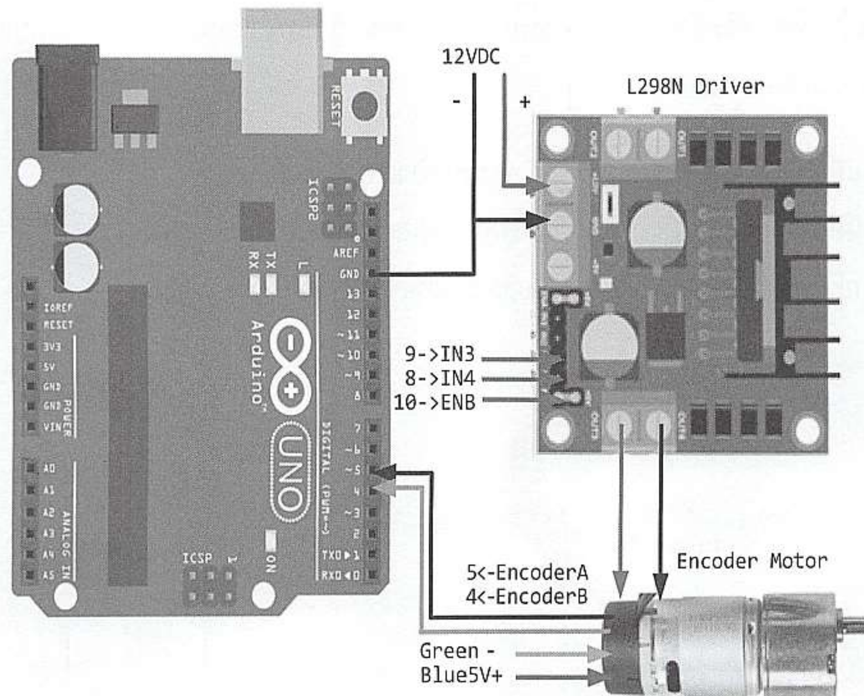


รูปที่ 4.10 เอนโค้ดเดอร์มอเตอร์

จากรูปที่ 4.10 เป็นมอเตอร์ของ Pololu รุ่น 2822 (Metal Gearmotor 37Dx68L mm with 64 CPR Encoder) เป็นดีซีมอเตอร์แบบมีเฟืองทด มีวงจรเอนโค้ดเดอร์ที่มีความละเอียดสูงสุด 64 พัลส์ต่อการหมุน 1 รอบ ใช้แรงดันไฟฟ้า 12 โวลต์ กระแสไฟฟ้า 300 มิลลิแอมแปร์ โดยมีสายสัญญาณ 6 เส้นดังนี้

สายสีแดง	คือสายสัญญาณไฟ 12 โวลต์หรือกราวด์ของมอเตอร์
สายสีดำ	คือสายสัญญาณไฟ 12 โวลต์หรือกราวด์ของมอเตอร์
สายสีเขียว	คือกราวด์ของวงจรเอนโค้ดเดอร์
สายสีน้ำเงิน	คือสัญญาณไฟของวงจรเอนโค้ดเดอร์ (3.5–20 โวลต์)
สายสีเหลือง	คือสัญญาณพัลส์เอาต์พุต A
สายสีขาว	คือสัญญาณพัลส์เอาต์พุต B

4.4 วงจรควบคุมเอนโค้ดเดอร์มอเตอร์



รูปที่ 4.11 วงจรควบคุมเอนโค้ดเดอร์มอเตอร์

จากรูปที่ 4.11 สัญญาณเอาต์พุตขา 8 9 และ 10 ต่อเข้ากับบอร์ดขับมอเตอร์ L298N ขา IN4 IN3 และขา ENB เพื่อควบคุมทิศทางและความเร็วของมอเตอร์ แล้วส่งสัญญาณเอาต์พุตออกไปควบคุมเอนโค้ดเดอร์มอเตอร์ให้หมุนตามทิศทางและความเร็วที่กำหนด ขาสัญญาณพัลส์เอนโค้ดเดอร์ A ต่อกับขา 5 เอนโค้ดเดอร์ B ต่อกับขา 4 และต้องต่อไฟเลี้ยงกับกราวด์ให้วงจร การควบคุมความเร็วของมอเตอร์ต้องส่งสัญญาณ PWM ออกทางขา 10 ที่ต่อเข้ากับขา ENB

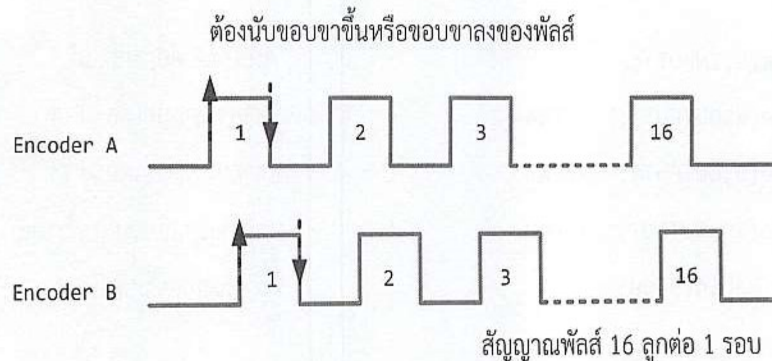
ตัวอย่าง

analogWrite(10,255); หมายถึงให้มอเตอร์หมุนด้วยความเร็วสูงสุด

analogWrite(10,127); หมายถึงส่งสัญญาณ PWM 50% ให้กับมอเตอร์

4.5 การตรวจนับขอบขาขึ้นและขอบขาลงของสัญญาณพัลส์

การนับสัญญาณเอนโค้ดเดอร์หรือสัญญาณพัลส์ต้องนับที่ขอบขาขึ้นหรือขอบขาลง โดยเอนโค้ดเดอร์เอาต์พุต A และ B จะให้สัญญาณพัลส์อย่างละ 16 พัลส์ ทำให้นับสัญญาณที่ขอบขาขึ้นและขาลงได้ละเอียดสุด 64 ครั้งต่อการหมุน 1 รอบ



รูปที่ 4.12 สัญญาณเอนโค้ดเดอร์ A และ B

ตัวอย่างการเขียนโปรแกรมนับขอบขาขึ้นของสัญญาณพัลส์

```
if(pulseA=digitalRead(5)==0)        //ถ้า pulseA เป็น 0
  {
    stateA=1;                        //ให้ตัวแปร stateA = 1
  }
if(pulseA=digitalRead(5)==1&&stateA==1)   //ตรวจสอบขอบขาขึ้นของสัญญาณพัลส์
{
  Count=Count+1;                     //นับสัญญาณ
  stateA=0;                          //ให้ตัวแปร stateA = 0
}
```

ตัวอย่างที่ 4.6 โปรแกรมนับสัญญาณพัลส์ A

```
int state1,count,pulseA;                   //ประกาศตัวแปร
void setup()
{
  pinMode(5,INPUT);                        //pulseA ต่อกับขา 5
  pinMode(8,OUTPUT);  //IN4                //ขาควบคุมมอเตอร์ IN4
  pinMode(9,OUTPUT);  //IN3                //ขาควบคุมมอเตอร์ IN3
  pinMode(10,OUTPUT); //PWM                //ขาสัญญาณควบคุมความเร็ว
  Serial.begin(9600);                      //กำหนดอัตราการสื่อสารแบบอนุกรม
}
void loop()
{
  analogWrite(10,127);                     //กำหนดความเร็ว PWM = 127
  digitalWrite(8,LOW);                     //ขา 8 เป็น LOW
  digitalWrite(9,HIGH);                    //ขา 9 เป็น HIGH มอเตอร์หมุน
  if(pulseA=digitalRead(5)==0)             //ถ้าขา 5 หรือ pulseA เป็น 0
    state1=1;
if(pulseA=digitalRead(5)==1&&state1==1)    //ถ้าเป็นขอบขาขึ้น
  {
    count++;                               //นับสัญญาณพัลส์
    state1=0;
  } Serial.println(count);                 //แสดงค่าการนับสัญญาณพัลส์
}
```

ผลการรันโปรแกรม

เมื่อมอเตอร์หมุน โปรแกรมจะนับสัญญาณเอนโค้ดเดอร์หรือสัญญาณพัลส์เอาต์พุต A ขอบขาขึ้น แล้วแสดงผลบนจอภาพผ่านพอร์ตอนุกรม (Serial Monitor)

ตัวอย่างที่ 4.7 โปรแกรมนับสัญญาณพัลส์ A ขอบขาขึ้นและขอบขาลง

```
int state1,state2,count,pulseA;
void setup()
{
  pinMode(5,INPUT); //pulseA
  pinMode(8,OUTPUT); //IN4
  pinMode(9,OUTPUT); //IN3
  pinMode(10,OUTPUT); //PWM
  Serial.begin(9600);
}
void loop()
{
  analogWrite(10,200);  //PWM=200
  digitalWrite(8,LOW);  //pin8=LOW
  digitalWrite(9,HIGH); //pin9=HIGH Motor On
  if(pulseA=digitalRead(5)==1)
    state1=1;
  if(pulseA=digitalRead(5)==0&&state1==1) //0->1
  {
    count++;
    state1=0;
  }
  if(pulseA=digitalRead(5)==0)  state2=1;
  if(pulseA=digitalRead(5)==1&&state2==1) //1->0
  {
    count++;
    state2=0;
  }
  Serial.println(count);
}
```

ผลการรันโปรแกรม

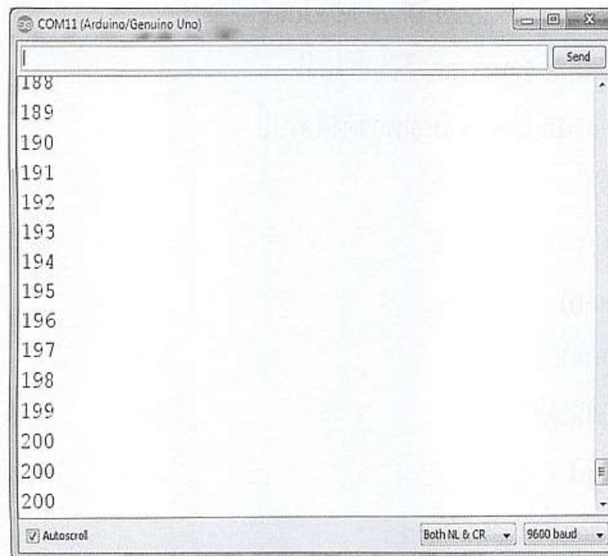
เมื่อมอเตอร์หมุน โปรแกรมจะนับสัญญาณเอนโค้ดเดอร์หรือสัญญาณพัลส์เอาต์พุต A ขอบขาขึ้นและขอบขาลง แล้วแสดงผลบนจอภาพผ่านพอร์ตอนุกรม (Serial Monitor)

ตัวอย่างที่ 4.8 โปรแกรมนับสัญญาณพัลส์ B 200 ลูกตามการกดสวิตช์ที่ขา 6

```
int state1,count,count1,pulseB;                    //ประกาศตัวแปร
void setup()
{ pinMode(4,INPUT);     //pulseB                   //ขา 4 รับสัญญาณพัลส์ B
  pinMode(6, INPUT);    //SW_Run                   //ขา 6 สวิตช์สั่งงาน
  pinMode(8, OUTPUT);  //IN4                       //ขาควบคุมมอเตอร์ IN4
  pinMode(9, OUTPUT);  //IN3                       //ขาควบคุมมอเตอร์ IN3
  pinMode(10, OUTPUT); //PWM                       //ขาควบคุมความเร็วมอเตอร์
  Serial.begin(9600);                              //กำหนดอัตราการสื่อสารแบบอนุกรม
}
void loop()
{
  if(digitalRead(6)==0)                            //ถ้าสวิตช์ขา 6 ถูกกด
  { analogWrite(10,100);                           //กำหนดความเร็ว PWM = 100
    digitalWrite(8,LOW);                           //ขา 8 เป็น LOW
    digitalWrite(9,HIGH);                          //ขา 9 เป็น HIGH มอเตอร์หมุน
    count=0;
  }
if(count<200)                                      //ถ้า count < 200
  {
    if(pulseB=digitalRead(5)==1)                   //ถ้าขา 5 หรือ pulseB เป็น 1
      state1=1;                                    //ให้ state1 = 1
  if(pulseB=digitalRead(5)==0&&state1==1)          //ถ้าเป็นพัลส์ขอบขาขึ้น
    { count++;                                     //นับสัญญาณพัลส์
      state1=0;
    }
  }
Serial.println(count);                             //แสดงผลการนับบนจอภาพ
if(count>=200)                                     //ถ้า count มากกว่าหรือเท่ากับ 200
digitalWrite(9,LOW);                               //ให้มอเตอร์หยุดหมุน
}
```

ผลการรันโปรแกรม

เมื่อกดสวิตช์ที่ขา 6 มอเตอร์หมุน โปรแกรมจะนับสัญญาณเอนโค้ดเดอร์หรือสัญญาณพัลส์เอาต์พุต B ขอบขาขึ้นจำนวน 200 ครั้ง แล้วจึงหยุดการทำงานของมอเตอร์

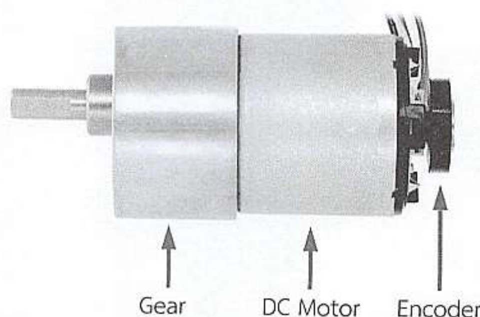


รูปที่ 4.13 นับสัญญาณเอนโค้ดเดอร์ครบ 200 ครั้งแล้วหยุด

4.6 สรุป

ดีซีมอเตอร์ หรือมอเตอร์ไฟฟ้ากระแสตรงเป็นเครื่องกลไฟฟ้าที่ทำหน้าที่เปลี่ยนพลังงานไฟฟ้าให้เป็นพลังงานกล เคลื่อนที่โดยการหมุน ดีซีมอเตอร์มีโครงสร้างหลักที่สำคัญ 3 ส่วนคือ แม่เหล็กถาวรที่ติดรอบตัวถังของมอเตอร์ แกนเหล็ก และขดลวดทองแดงที่พันรอบแกนเหล็ก โดยใช้บอร์ด Motor Drive Module L298N เป็นตัวขับ ซึ่งจะควบคุมทิศทางการหมุนและความเร็วของมอเตอร์ได้

เอนโค้ดเดอร์มอเตอร์ คือดีซีมอเตอร์ที่มีเฟืองทดรอบเพื่อเพิ่มแรงบิด และมีวงจรเอนโค้ดเดอร์ที่ใช้ตรวจวัดจำนวนรอบในการหมุนของมอเตอร์เพื่อควบคุมหรือนับจำนวนรอบการหมุนของมอเตอร์ เอนโค้ด-เดอร์มี 3 ส่วนหลัก ส่วนที่ 1 คือชุดเฟือง (Gear) ทำหน้าที่ทดรอบเพิ่มแรงบิดให้มอเตอร์ ส่วนที่ 2 คือดีซีมอเตอร์ ซึ่งเป็นมอเตอร์ไฟฟ้ากระแสตรง และส่วนที่ 3 คือเอนโค้ดเดอร์ เป็นวงจรนับสัญญาณพัลส์หรือสัญญาณเอนโค้ดเดอร์



รูปที่ 4.14 ส่วนประกอบของเอนโค้ดเดอร์มอเตอร์

แบบฝึกหัดบทที่ 4

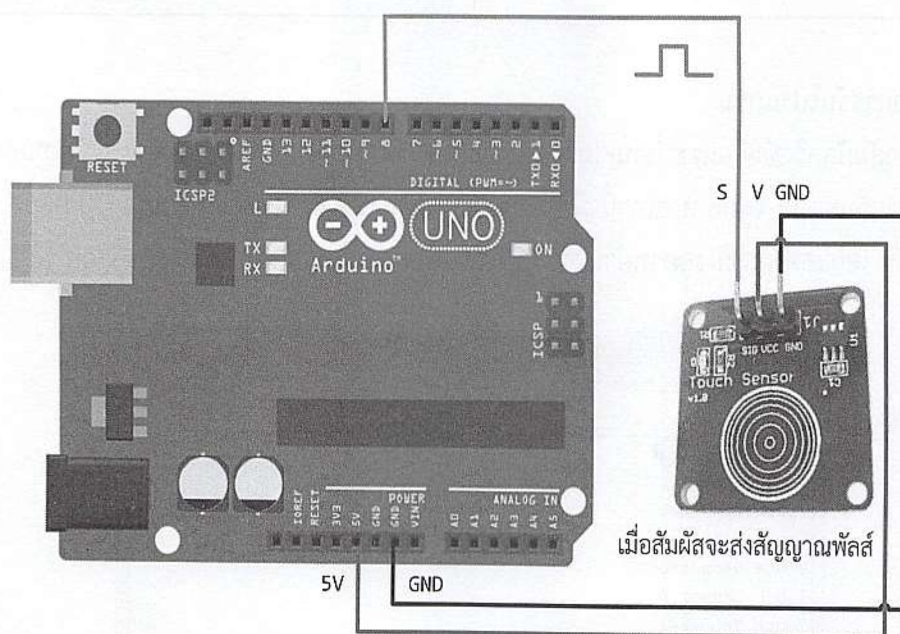
1. อธิบายขาสัญญาณของบอร์ด Motor Drive Module L298N
2. ให้เขียนโปรแกรมควบคุมความเร็วของดีซีมอเตอร์
3. อธิบายขาสัญญาณของเอนโค้ดเดอร์มอเตอร์ดังต่อไปนี้
   - สายสีแดง (Red)
   - สายสีดำ (Black)
   - สายสีเขียว (Green)
   - สายสีน้ำเงิน (Blue)
   - สายสีเหลือง (Yellow)
   - สายสีขาว (White)
4. อธิบายการตรวจนับสัญญาณเอนโค้ดเดอร์
5. ให้เขียนโปรแกรมควบคุมเอนโค้ดเดอร์ให้นับ 100 ครั้งตามการกดสวิตช์
6. ให้เขียนโปรแกรมควบคุมเอนโค้ดเดอร์ให้หมุนซ้ายหรือขวาตามการกดสวิตช์

บทที่ 5 เซนเซอร์

เซนเซอร์ (Sensor) คืออุปกรณ์ตรวจจับสัญญาณทางฟิสิกส์ เช่น อุณหภูมิ ความชื้น เสียง แสง การสัมผัส หรือการเคลื่อนไหว เซนเซอร์ในปัจจุบันสามารถเชื่อมต่อและเขียนโปรแกรมใช้งานได้สะดวก มีไลบรารีให้ใช้งานเพื่อให้สามารถอ่านค่าของเซนเซอร์และพัฒนาระบบควบคุมได้โดยง่าย

5.1 เซนเซอร์แบบสัมผัส

เซนเซอร์แบบสัมผัส (Touch Sensor) ใช้แทนการทำงานของสวิตช์ปกติ ซึ่งจะทำงานเมื่อมีการแตะหรือสัมผัส (ไม่ใช่เซนเซอร์ตรวจสอบลายนิ้วมือ) เซนเซอร์ชนิดนี้มี 3 ขา คือ ขากราวด์ ขาไฟบวก 5 โวลต์ และขาสัญญาณเอาต์พุตซึ่งต่อกับขา 8 ดังรูปที่ 5.1



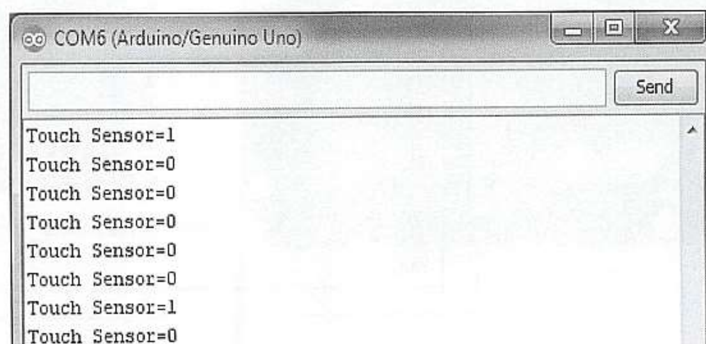
รูปที่ 5.1 วงจรการเชื่อมต่อ Arduino กับเซนเซอร์แบบสัมผัส

ตัวอย่างที่ 5.1 โปรแกรมตรวจจับเซนเซอร์แบบสัมผัส

```
void setup()
{ Serial.begin(9600);              //กำหนดการสื่อสารแบบอนุกรม 9600 b/s
  pinMode(13,OUTPUT);              //กำหนดขา 13 เป็นเอาต์พุต
  pinMode(8,INPUT);                //กำหนดขา 8 เป็นอินพุต
}
void loop()
{
  int Tsensor=digitalRead(8);      //อ่านค่าเซนเซอร์จากขา 8
  Serial.print("Touch Sensor=");   //แสดงข้อความ
  Serial.println(Tsensor);         //แสดงค่าของเซนเซอร์แบบสัมผัส
  if(Tsensor==HIGH)                //ถ้าเซนเซอร์ถูกสัมผัสให้ทำงาน
  {
    digitalWrite(13,HIGH);         //ขา 13 เป็น HIGH ไฟบนบอร์ดติด
    delay(300) ;                   //หน่วงเวลา 300 ms
  }
  digitalWrite(13,LOW);            //ขา 13 เป็น LOW ไฟบนบอร์ดดับ
  delay(100);
}
```

ผลการรันโปรแกรม

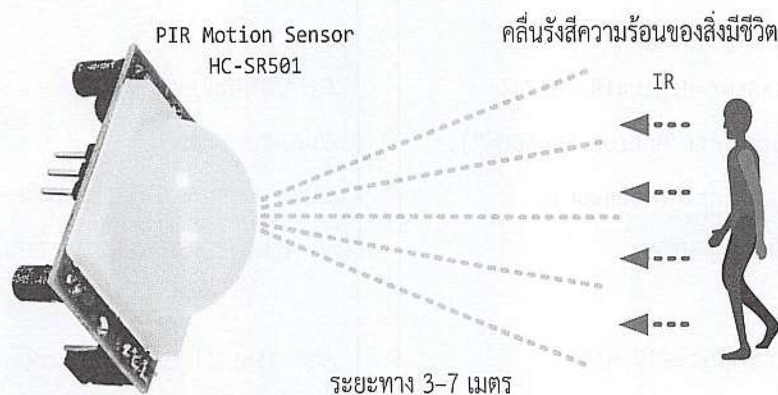
เมื่อสัมผัสสวิตช์หรือแตะที่เซนเซอร์ สัญญาณที่ขา 8 ทำงาน Touch Sensor = 1 หลอด LED บนบอร์ดที่ต่อกับขา 13 จะติด หากเอานิ้วมือออกหรือยกเลิกการสัมผัส Touch Sensor = 0 หลอด LED ที่ขา 13 จะดับ โดยแสดงผลบนจอภาพผ่านการสื่อสารแบบอนุกรมโดยใช้เมนู Serial Monitor



รูปที่ 5.2 การแสดงผลของเซนเซอร์แบบสัมผัสผ่าน Serial Monitor

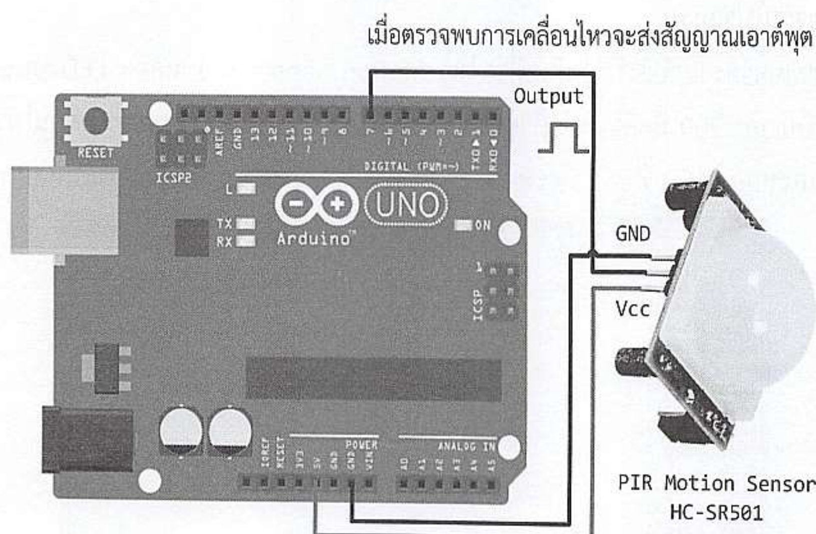
5.2 เซนเซอร์ตรวจจับความเคลื่อนไหว

PIR (Passive Infrared) Motion Sensor คือเซนเซอร์ตรวจจับความเคลื่อนไหวจากการเปลี่ยนแปลงของคลื่นรังสีความร้อนหรือคลื่นอินฟราเรดจากสิ่งมีชีวิต เช่น เมื่อมีคนเดินผ่าน เซนเซอร์ก็สามารถตรวจจับการเคลื่อนไหวจากรังสีความร้อนที่เปลี่ยนแปลงได้ แล้วส่งสัญญาณดิจิทัลออกมาทางขาสัญญาณเอาต์พุต เซนเซอร์ HC-SR501 เป็นเซนเซอร์ตรวจจับความเคลื่อนไหวที่มีระยะตรวจจับ 3–7 เมตร มุมการตรวจจับน้อยกว่า 140 องศา และใช้เวลาในการตรวจจับ 5–200 วินาที



รูปที่ 5.3 เซนเซอร์ตรวจจับความเคลื่อนไหว

อินฟราเรด (Infrared) หรือคลื่นใต้แดง เป็นคลื่นแม่เหล็กไฟฟ้าชนิดหนึ่งที่มีความยาวคลื่น 700 นาโนเมตรถึง 1 มิลลิเมตร เป็นคลื่นรังสีความร้อนชนิดหนึ่งที่มนุษย์ไม่สามารถมองเห็นแต่รู้สึกถึงความร้อนได้



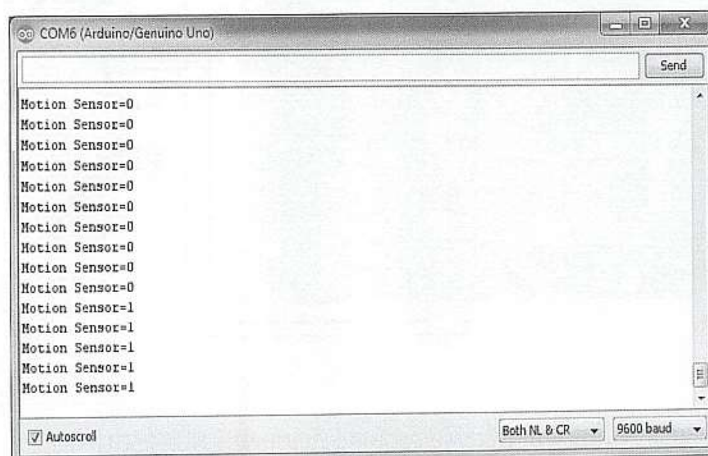
รูปที่ 5.4 การเชื่อมต่อ Arduino กับเซนเซอร์ HC-SR501

ตัวอย่างที่ 5.2 โปรแกรมตรวจจับความเคลื่อนไหว

```
void setup()
{ Serial.begin(9600);            //กำหนดการสื่อสารแบบอนุกรม 9600 b/s
  pinMode(13,OUTPUT);            //กำหนดขา 13 เป็นเอาต์พุต
  pinMode(7,INPUT);              //กำหนดขา 7 เป็นอินพุต
}
void loop()
{
  int Msensor=digitalRead(7);    //อ่านค่าเซนเซอร์จากขา 7
  Serial.print("Motion Sensor=");  //แสดงข้อความ
  Serial.println(Msensor);       //แสดงค่าเซนเซอร์ตรวจจับความเคลื่อนไหว
  if(Msensor==HIGH)              //ถ้าเซนเซอร์ตรวจพบความเคลื่อนไหว
  {
    digitalWrite(13,HIGH);       //ขา 13 เป็น HIGH ไฟบนบอร์ดติด
    delay(200) ;
  }
  digitalWrite(13,LOW);          //ขา 13 เป็น LOW ไฟบนบอร์ดดับ
  delay(100);
}
```

ผลการรันโปรแกรม

เมื่อเซนเซอร์ตรวจจับสิ่งมีชีวิตที่เคลื่อนไหว Motion Sensor = 1 หลอด LED บนบอร์ดที่ต่อกับขา 13 จะติดเป็นเวลา 200 มิลลิวินาทีแล้วดับ หากเซนเซอร์ไม่สามารถตรวจจับการเคลื่อนไหวได้ Motion Sensor = 0 และหลอด LED ที่ขา 13 จะดับ โดยแสดงผลบนจอภาพผ่านการสื่อสารแบบอนุกรม

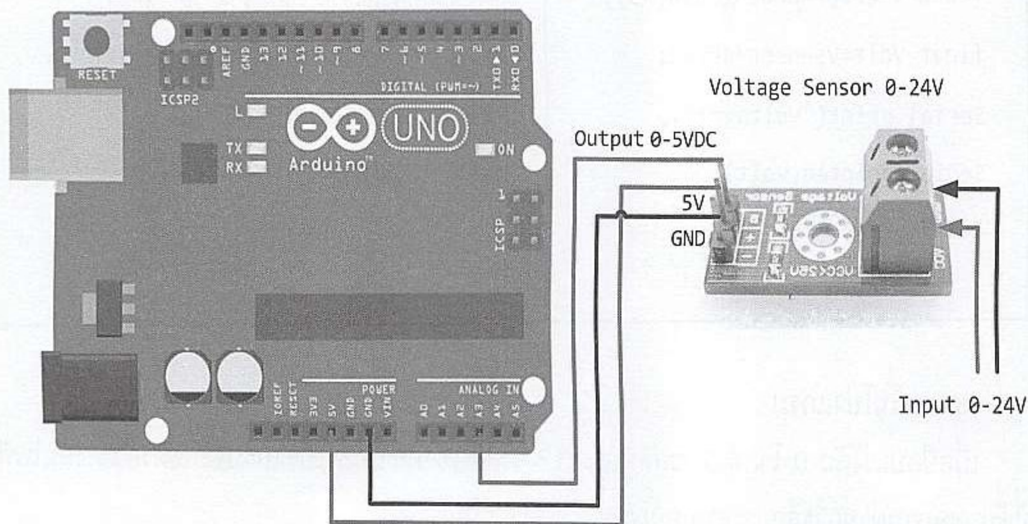


รูปที่ 5.5 การแสดงผลการตรวจจับความเคลื่อนไหวผ่าน Serial Monitor

5.3 เซนเซอร์วัดแรงดันไฟฟ้า

เซนเซอร์วัดแรงดันไฟฟ้าสามารถวัดแรงดันไฟฟ้ากระแสตรงขนาด 0–24 โวลต์ ตัวเซนเซอร์มีขาสัญญาณ 3 ขา คือ ขากราวด์ ขาสัญญาณไฟ และขาสัญญาณเอาต์พุตแอนะล็อก 0–5 โวลต์ ซึ่งจะแปลงเป็นสัญญาณดิจิทัล 10 บิตหรือ 0–1023 ระดับ

ในการเชื่อมต่อกับบอร์ด Arduino จะต่อขาสัญญาณเอาต์พุตของเซนเซอร์เข้าที่ขา A3 ของบอร์ดและต่อแรงดันไฟฟ้าที่ต้องการวัดซึ่งต้องอยู่ในช่วง 0–24 โวลต์เข้าขาอินพุตของเซนเซอร์ ในการทดสอบจะใช้แรงดัน 3.3 โวลต์ และ 5 โวลต์ของบอร์ดมาใช้งาน



รูปที่ 5.6 การเชื่อมต่อ Arduino กับเซนเซอร์วัดแรงดัน

การเขียนโปรแกรมตรวจวัดแรงดันไฟฟ้าจะประกาศตัวแปรแบบทศนิยม (Float) และค่าที่ได้ต้องนำมาหารด้วย 40.92 เพื่อแปลงค่า 0–1023 ให้เป็น 0–25 (1023/25 = 40.92) ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

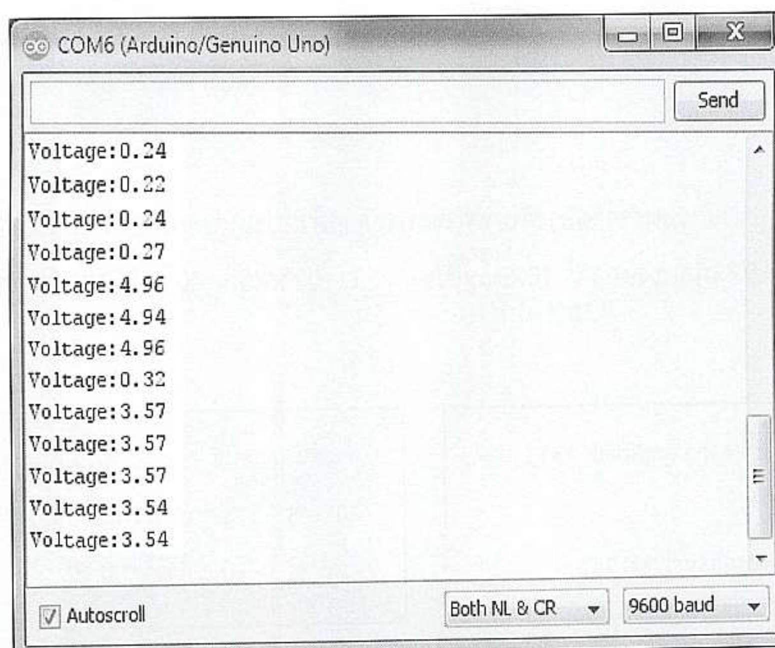
```
float Vsensor=analogRead(A3);    //ประกาศตัวแปร Vsensor เป็นแบบทศนิยม
                                 //และอ่านค่าจากขาสัญญาณ A3 มีค่า 0–1023
float Volt=Vsensor/40.92;        //แปลงค่า 0–1023 ให้เป็น 0–25
```

ตัวอย่างที่ 5.3 โปรแกรมตรวจวัดแรงดันไฟฟ้า

```
void setup()
{
  Serial.begin(9600);                    //กำหนดการสื่อสารแบบอนุกรม 9600 b/s
}
void loop()
{
  float Vsensor=analogRead(A3);          //อ่านค่าสัญญาณจากขา A3 มีค่า 0–1023
  float Volt=Vsensor/40.92;              //แปลงเป็นค่าแรงดันไฟฟ้าให้อยู่ในช่วง 0–25 V
  Serial.print("Voltage:");              //แสดงข้อความ
  Serial.println(Volt);                  //แสดงค่าแรงดันไฟฟ้าที่วัดได้
  delay(500);
}
```

ผลการรันโปรแกรม

เมื่อป้อนแรงดัน 0 โวลต์ 5 โวลต์ และ 3.3 โวลต์ เข้าที่ขา A3 โปรแกรมจะแสดงค่าแรงดันไฟฟ้าที่วัดได้บนจอภาพผ่านการสื่อสารแบบอนุกรม



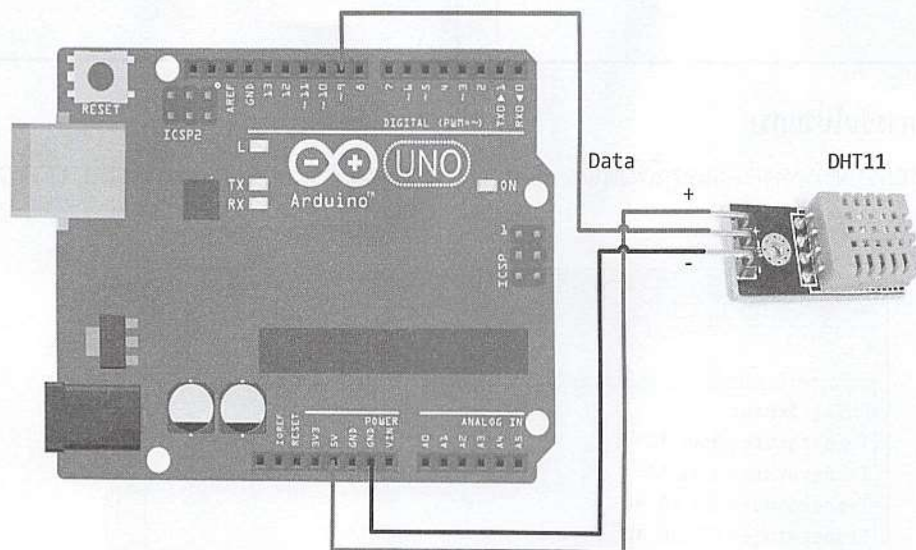
รูปที่ 5.7 แสดงผลการตรวจวัดแรงดันไฟฟ้า

5.4 เซนเซอร์วัดอุณหภูมิและความชื้น

DHT (Digital Humidity and Temperature Sensor) เป็นเซนเซอร์สำหรับวัดความชื้นและอุณหภูมิในอากาศ สามารถวัดความชื้นสัมพัทธ์ (RH : Relative Humidity) ได้ในช่วง 20–90% มีค่าผิดพลาด ±5% วัดอุณหภูมิได้ 0–50 องศาเซลเซียส มีค่าผิดพลาด ±2 องศาเซลเซียส

ในการเชื่อมต่อวงจรจะต่อขาสัญญาณของเซนเซอร์เข้ากับขาดิจิทัลขา 9 ของบอร์ด Arduino การสื่อสารรับและส่งข้อมูลระหว่างเซนเซอร์ DHT และ Arduino จะสื่อสารข้อมูลแบบอนุกรมโดยใช้สายสัญญาณเพียงเส้นเดียว ใช้เวลาในการอ่านค่าสัญญาณ (Sample Rate) ทุก ๆ 1 วินาที ในการใช้งานต้องเรียกใช้ไลบรารี DHT.h (include <DHT.h>) จึงจะสามารถใช้ฟังก์ชันในการอ่านค่าความชื้นและอุณหภูมิได้โดยง่าย ตัวอย่างการเขียนโปรแกรมมีดังนี้

```
#include <DHT.h>                              //เรียกใช้ไลบรารี DHT.h
DHT dht(9,DHT11);                             //กำหนดขา 9 ต่อกับ DHT11
float t = dht.readTemperature();              //อ่านค่าอุณหภูมิมาเก็บไว้ในตัวแปร t
float h = dht.readHumidity();                 //อ่านค่าความชื้นมาเก็บไว้ในตัวแปร h
```



รูปที่ 5.8 การเชื่อมต่อ Arduino กับเซนเซอร์ DHT11

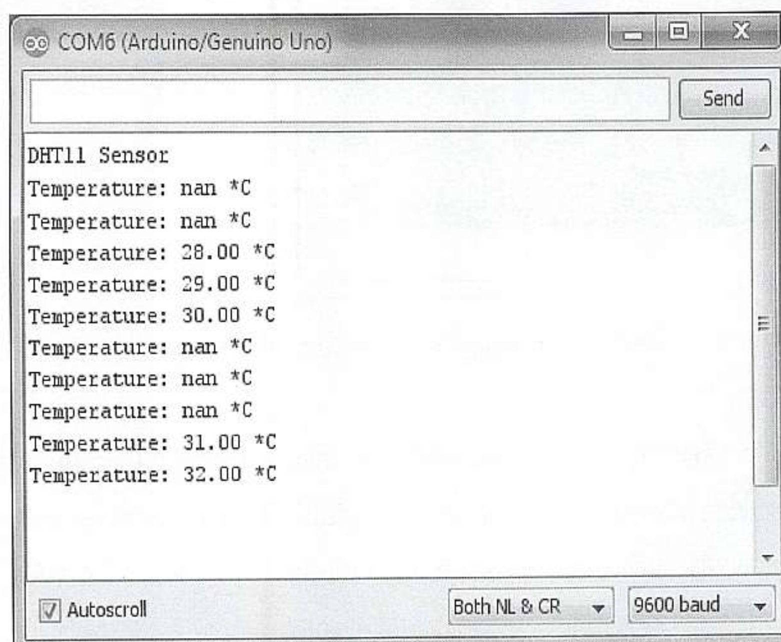
DHT11 มีการสื่อสารแบบ 1-Wire ซึ่งมีข้อดีคือใช้สายสัญญาณเพียงเส้นเดียวในการรับและส่งข้อมูล การรับ-ส่งข้อมูลจะใช้ช่วงเวลาที่ต่างกัน (Half-duplex) หรือไม่สามารถรับและส่งข้อมูลในเวลาเดียวกันได้ นอกจากนี้ลักษณะการทำงานยังเป็นแบบ Master/Slave โดยที่อุปกรณ์ Master หรือไมโครคอนโทรลเลอร์จะเป็นตัวควบคุมสถานะการทำงานและจังหวะการรับ-ส่งของบัสข้อมูล

ตัวอย่างที่ 5.4 โปรแกรมตรวจวัดอุณหภูมิ

```
#include <DHT.h>                          //เรียกใช้ไลบรารี DHT.h
DHT dht(9,DHT11);                         //กำหนดขา 9 ต่อกับ DHT11
void setup()
{
  Serial.begin(9600);
  Serial.println("DHT11 Sensor");
  dht.begin();                            //เริ่มการทำงานของเซนเซอร์
}
void loop()
{
  delay(1000);                            //หน่วงเวลา 1000 ms
  float t = dht.readTemperature();        //อ่านค่าอุณหภูมิมาเก็บไว้ในตัวแปร t
  Serial.print("Temperature: ");          //แสดงข้อความ
  Serial.print(t);                        //แสดงค่าอุณหภูมิบนจอภาพ
  Serial.print(" *C\n");
}
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงค่าอุณหภูมิที่วัดได้บนจอภาพผ่านการสื่อสารแบบอนุกรม ซึ่งจะมีบางช่วงเวลาที่ไม่สามารถอ่านค่าได้



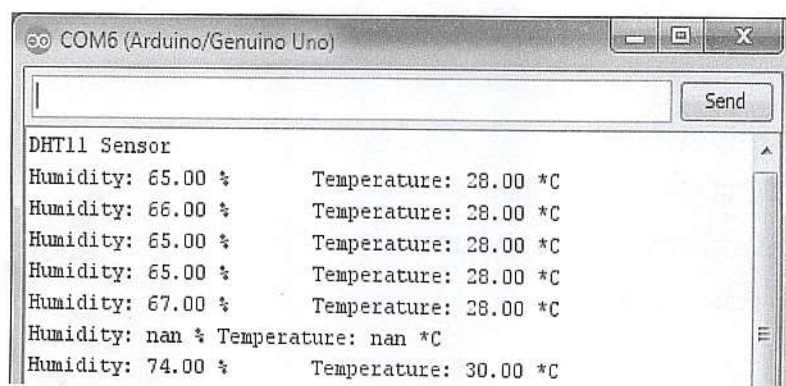
รูปที่ 5.9 แสดงผลการตรวจวัดอุณหภูมิ

ตัวอย่างที่ 5.5 โปรแกรมตรวจวัดความชื้นและอุณหภูมิ

```
#include <DHT.h>                            //เรียกใช้ไลบรารี DHT.h
DHT dht(9,DHT11);                           //กำหนดขา 9 ต่อกับ DHT11
void setup()
{ Serial.begin(9600);
  Serial.println("DHT11 Sensor");
  dht.begin();                              //เริ่มการทำงานของเซนเซอร์ DHT
}
void loop()
{ delay(1000);
  float h = dht.readHumidity();             //อ่านค่าความชื้นมาเก็บไว้ในตัวแปร h
  float t = dht.readTemperature();          //อ่านค่าอุณหภูมิมาเก็บไว้ในตัวแปร t
  Serial.print("Humidity: ");               //แสดงข้อความ Humidity:
  Serial.print(h);                          //แสดงค่าความชื้นบนจอภาพ
  Serial.print(" %\t");
  Serial.print("Temperature: ");            //แสดงข้อความ Temperature:
  Serial.print(t);                          //แสดงค่าอุณหภูมิบนจอภาพ
  Serial.print(" *C\n");
}
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงค่าความชื้นและอุณหภูมิที่วัดได้บนจอภาพผ่านการสื่อสารแบบอนุกรม ซึ่งอาจจะมีบางช่วงเวลาที่ไม่สามารถอ่านค่าได้

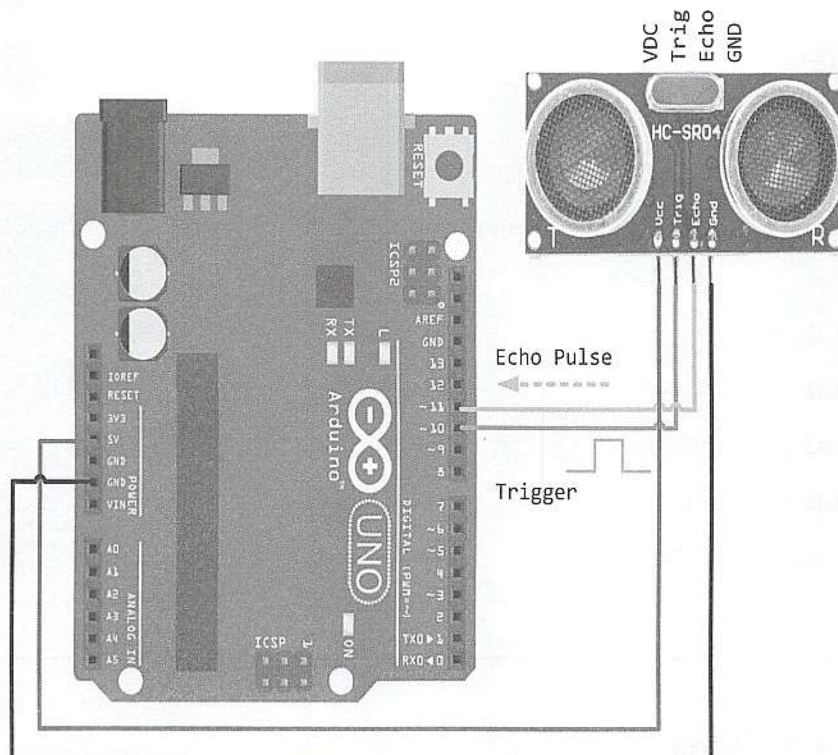


รูปที่ 5.10 แสดงผลการตรวจวัดความชื้นและอุณหภูมิ

5.5 เซนเซอร์วัดระยะทาง

HC-SR04 เป็นเซนเซอร์วัดระยะทางด้วยอัลตราโซนิก (Ultrasonic Sensor) หรือใช้คลื่นเสียงสะท้อนในการวัดระยะทาง ซึ่งสามารถวัดระยะได้ในช่วง 2–400 เซนติเมตร มีมุมในการวัด 15 องศา

หลักการทำงานคือ เมื่อส่งสัญญาณพัลส์ (Trig) 10 ไมโครวินาที เซนเซอร์จะส่งคลื่นความถี่ขนาด 40 กิโลเฮิรตซ์ออกมา 8 ลูกคลื่น เมื่อคลื่นเดินทางไปเจอวัตถุแล้วสะท้อนกลับมายังตัวรับ เซนเซอร์จะสามารถจับเวลาการเดินทางของคลื่นแล้วคำนวณระยะทางออกมาได้ โดยคำนวณจากสัญญาณสะท้อน (Echo Pulse) จากวงจร โดยการต่อวงจรจะต่อขา Trig เข้าขา 10 และขา Echo เข้าขา 11 ของบอร์ด Arduino



รูปที่ 5.11 การเชื่อมต่อ Arduino กับเซนเซอร์ HC-SR04

คุณลักษณะของเซนเซอร์ HC-SR04 Ultrasonic Module

- ความถี่ 40 กิโลเฮิรตซ์
- แรงดันไฟฟ้า 5 โวลต์
- ใช้กระแสไฟฟ้า 15 มิลลิแอมแปร์
- ระยะทางในการวัด 2–400 เซนติเมตร
- มุมการวัด 15 องศา

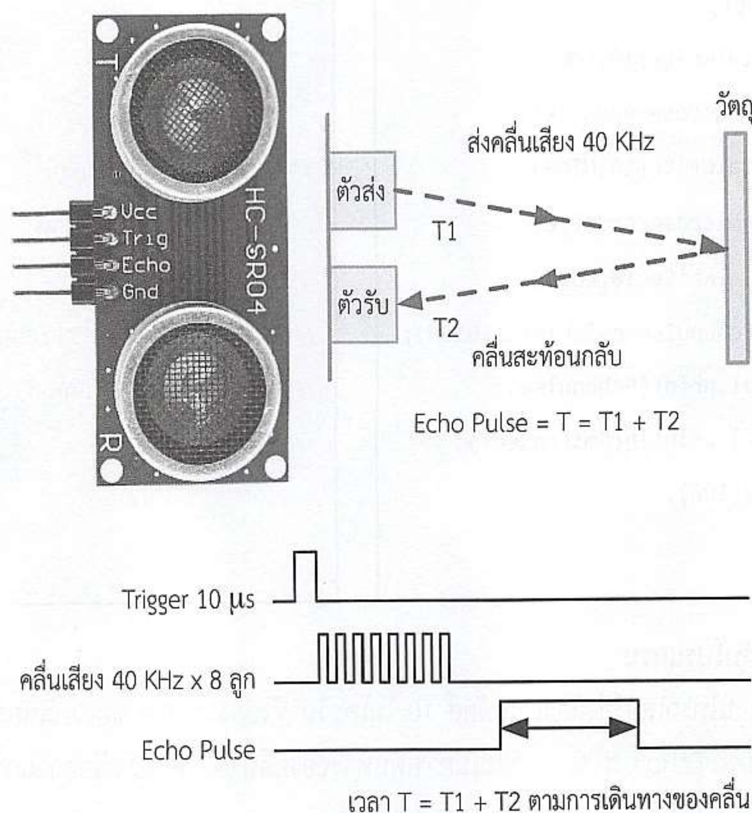
หลักการทำงานของเซนเซอร์วัดระยะทางด้วยอัลตราโซนิก

1) ไมโครคอนโทรลเลอร์ Arduino ส่งสัญญาณพัลส์ 10 ไมโครวินาทีไปยังขา Trig ของเซนเซอร์

2) เซนเซอร์กำเนิดสัญญาณเสียงความถี่ 40 กิโลเฮิรตซ์ จำนวน 8 ลูกส่งออกไปในอากาศ

3) ตัวรับรอรับสัญญาณแล้วส่งสัญญาณพัลส์ที่มีความกว้างตามเวลาการเดินทาง T = T1 + T2 ออกขา Echo

4) ไมโครคอนโทรลเลอร์คำนวณความกว้างของสัญญาณพัลส์เพื่อคำนวณเป็นระยะทาง



รูปที่ 5.12 หลักการทำงานของเซนเซอร์ HC-SR04

การคำนวณเวลาเป็นระยะทางในหน่วยเซนติเมตร

เสียงเดินทางด้วยความเร็ว 340 เมตรต่อวินาที หรือเท่ากับ

340 x 100 cm = 1 x 1,000,000 ไมโครวินาที

1 cm = 1,000,000/34,000 = 29.41 ไมโครวินาที

ดังนั้นระยะทางในหน่วยเซนติเมตร

= Echo Pulse / (29.41 x 2)

หรือ = Echo Pulse / 29.41 / 2

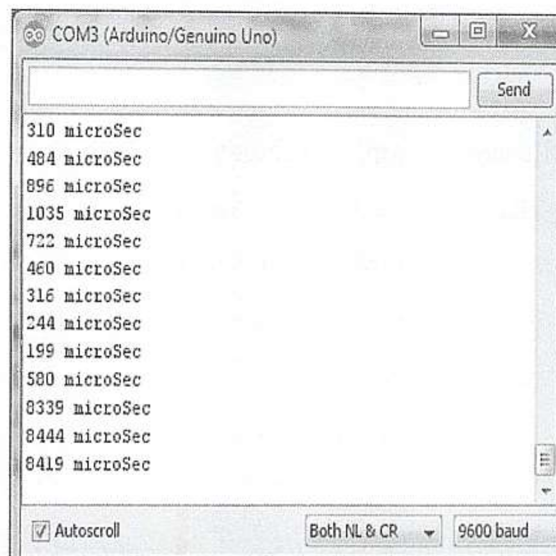
(หารด้วย 2 เพราะต้องการหาระยะทางเพียงครึ่งหนึ่งคือระยะทางไปหรือระยะทางกลับ)

ตัวอย่างที่ 5.6 การอ่านค่าเวลาการเดินทางของคลื่นจากเซนเซอร์

```
void setup()
  { Serial.begin(9600);                    //กำหนดการสื่อสารแบบอนุกรม 9600 b/s
    pinMode(11,INPUT);                     //กำหนดขา Echo
    pinMode(10,OUTPUT);                    //กำหนดขา Trig
  }
void loop()
  { digitalWrite(10,LOW);
    delayMicroseconds(10);
    digitalWrite(10,HIGH);                 //สร้างสัญญาณพัลส์ (Trig)
    delayMicroseconds(10);                 //หน่วงเวลา 10 ไมโครวินาที
    digitalWrite(10,LOW);
    int Echopulse=pulseIn(11,HIGH);        //อ่านสัญญาณพัลส์จากขา Echo
    Serial.print(Echopulse);               //แสดงเวลาของสัญญาณพัลส์
    Serial.println("microSec");            //แสดงข้อความ
    delay(100);
}
```

ผลการรันโปรแกรม

ไมโครคอนโทรลเลอร์ส่งสัญญาณพัลส์ 10 ไมโครวินาทีไปยังขา Trig ของเซนเซอร์ จากนั้นอ่านค่าสัญญาณพัลส์จากขา Echo ซึ่งเป็นเวลาที่ใช้ในการเดินทางของคลื่น (T1 + T2) หรือความกว้างของสัญญาณพัลส์มีหน่วยเป็นไมโครวินาที แล้วแสดงผลเวลาบนจอภาพ



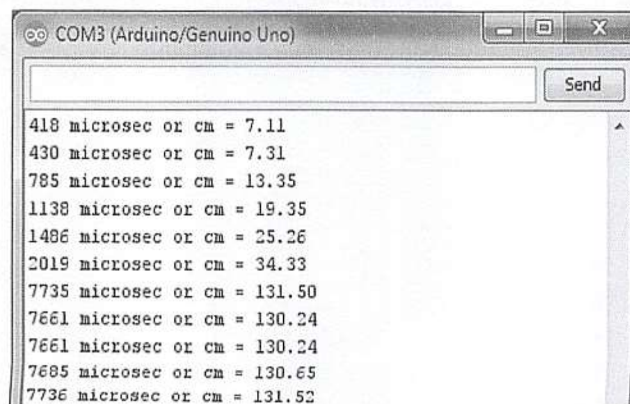
รูปที่ 5.13 การอ่านค่าเวลาในการเดินทางของคลื่น

ตัวอย่างที่ 5.7 โปรแกรมวัดระยะทางหน่วยเป็นเซนติเมตร

```
void setup()
{ Serial.begin(9600);
  pinMode(11,INPUT);                      //กำหนดขา Echo
  pinMode(10,OUTPUT);                     //กำหนดขา Trig
}
void loop()
{ digitalWrite(10,LOW);
  delayMicroseconds(1);
  digitalWrite(10,HIGH);                  //สร้างสัญญาณพัลส์ (Trig)
  delayMicroseconds(10);                  //หน่วงเวลา 10 ไมโครวินาที
  digitalWrite(10,LOW);
  int Echopulse=pulseIn(11,HIGH);         //อ่านสัญญาณพัลส์จากขา Echo
  float cm=Echopulse/(29.41*2);           //แปลงเป็นเซนติเมตร
  Serial.print(Echopulse);                //แสดงเวลาบนจอภาพ
  Serial.print(" microsec or cm =");      //แสดงข้อความ
  Serial.println(cm);                     //แสดงระยะทางบนจอภาพ
  delay(100);
}
```

ผลการรันโปรแกรม

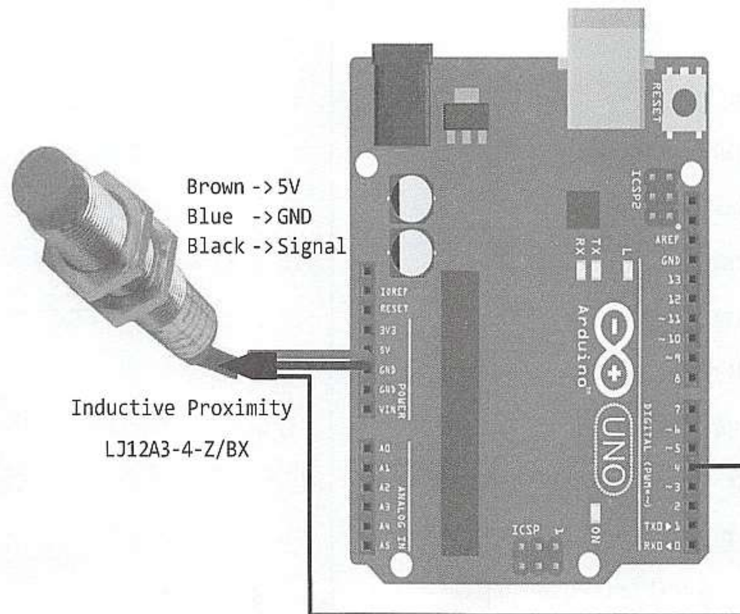
ไมโครคอนโทรลเลอร์ส่งสัญญาณพัลส์ 10 ไมโครวินาทีไปยังขา Trig ของเซนเซอร์ จากนั้นอ่านค่าสัญญาณพัลส์จากขา Echo เพื่อคำนวณหาระยะทาง หน่วยเป็นเซนติเมตร แล้วแสดงผลบนจอภาพทั้งเวลาในการเดินทางของคลื่นและระยะทางที่วัดได้



รูปที่ 5.14 การวัดระยะทางในหน่วยเซนติเมตร

5.6 เซนเซอร์ตรวจจับโลหะ

เซนเซอร์ตรวจจับโลหะหรือพร็อกซิมิตี้เซนเซอร์แบบอินดักทีฟ (Inductive Proximity Sensor) ใช้ในการตรวจจับโลหะ เช่น อะลูมิเนียม เหล็ก หรือโลหะอื่น ๆ มีระยะตรวจจับสูงสุด 4 มิลลิเมตร ใช้ไฟเลี้ยง 5–36 โวลต์ ใช้กระแสไฟฟ้า 300 มิลลิแอมแปร์ และให้สัญญาณเอาต์พุตแบบดิจิทัล



รูปที่ 5.15 การเชื่อมต่อ Arduino กับเซนเซอร์ตรวจจับโลหะ

การต่อสายและการทำงาน

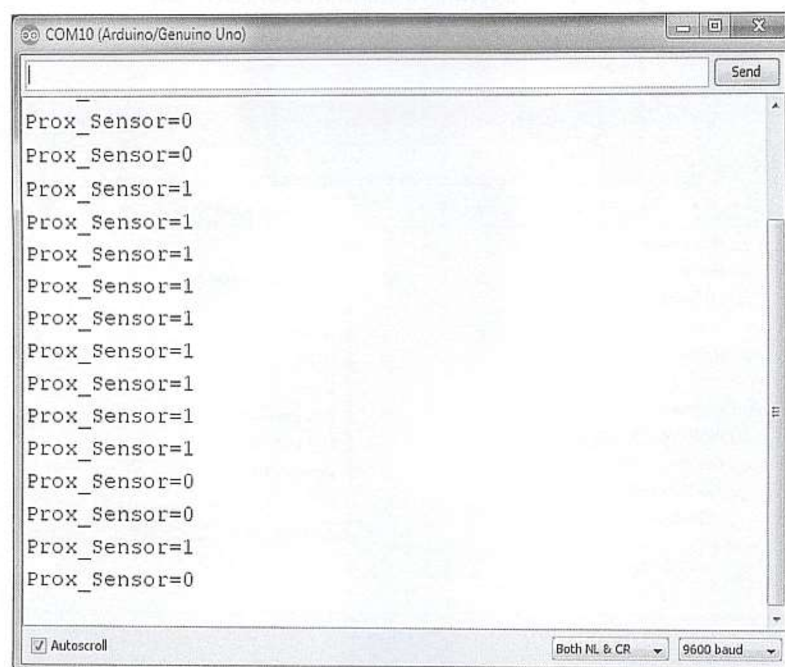
- สีน้ำตาล ต่อแรงดันไฟฟ้า 5–36 โวลต์
- สีน้ำเงิน ต่อกราวด์
- สีดำ สัญญาณเอาต์พุต
- เมื่อตรวจไม่พบวัตถุโลหะ LED บนเซนเซอร์จะดับ สัญญาณเอาต์พุตที่สายสีดำเป็นลอจิก 1
- เมื่อตรวจพบวัตถุโลหะ LED บนเซนเซอร์จะติด สัญญาณเอาต์พุตที่สายสีดำเป็นลอจิก 0

ตัวอย่างที่ 5.8 โปรแกรมตรวจจับโลหะ

```
void setup()                              //กำหนดการเริ่มต้น
{
  Serial.begin(9600);                     //กำหนดการสื่อสารแบบอนุกรม 9600 b/s
  pinMode(4,INPUT);                       //ให้ขา 4 เป็นอินพุตรับค่าจากเซนเซอร์
}
void loop()
{
  int ProxSensor=digitalRead(4);          //อ่านค่าเซนเซอร์จากขา 4
  Serial.print("Prox_Sensor=");           //แสดงข้อความ
  Serial.println(ProxSensor);             //แสดงค่าเซนเซอร์บนจอภาพ
  delay(100);                             //หน่วงเวลา
}
```

ผลการรันโปรแกรม

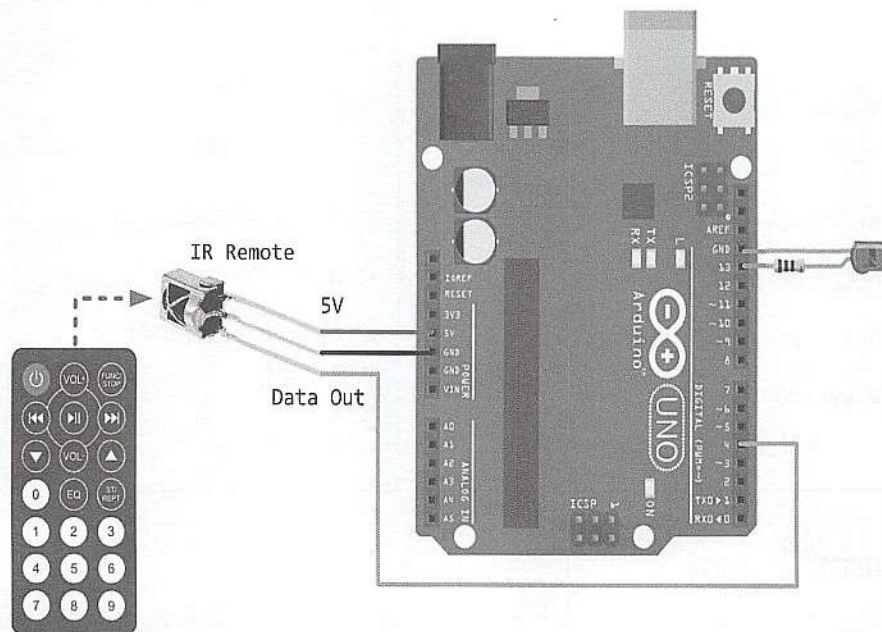
โปรแกรมจะแสดงค่าลอจิก 0 หรือลอจิก 1 ของเซนเซอร์ตรวจจับโลหะจากขา 4 แล้วแสดงผลบนจอภาพผ่านการสื่อสารแบบอนุกรม



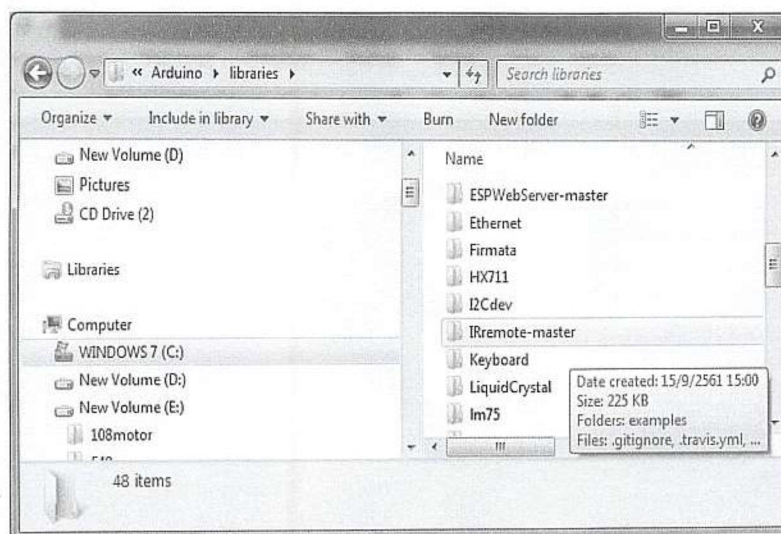
รูปที่ 5.16 ผลการรันโปรแกรมตรวจจับโลหะ

5.7 รีโมทอินฟราเรด

รีโมทอินฟราเรด (Infrared Remote) คือตัวควบคุมระยะไกลโดยใช้คลื่นรังสีอินฟราเรดในการรับและส่งสัญญาณ ซึ่งจะมีตัวส่งสัญญาณคือรีโมทและตัวรับสัญญาณ ตัวรับสัญญาณมีสัญญาณ 3 ขาคือ ไฟเลี้ยง กราวด์ และขาข้อมูลที่ต่อเข้ากับขา 4 ของบอร์ด การเขียนโปรแกรมต้องใช้ไลบรารี #include <IRremote.h> และติดตั้งไฟล์ IRremote ไว้ในไลบรารีของโปรแกรม



รูปที่ 5.17 การเชื่อมต่อ Arduino กับรีโมทอินฟราเรด



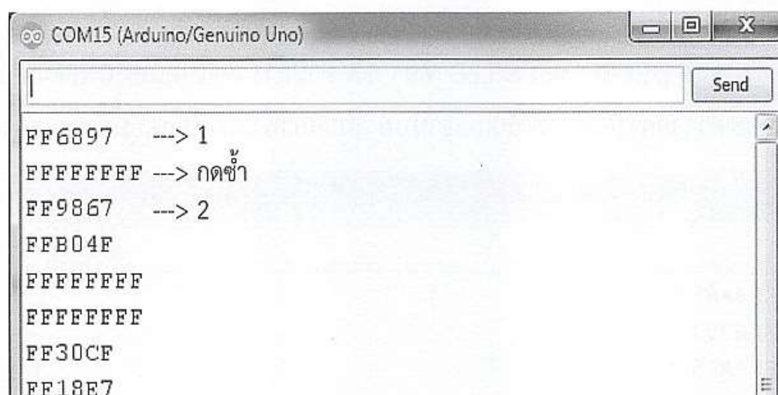
รูปที่ 5.18 การติดตั้งไฟล์ IRremote ไว้ในไลบรารี

ตัวอย่างที่ 5.9 โปรแกรมอ่านค่าจากรีโมทอินฟราเรด

```
#include <IRremote.h>                         //เรียกใช้ไลบรารี IRremote.h
int RECV_PIN = 6;                             //ให้ RECV_PIN = 6
IRrecv irrecv(RECV_PIN);                      //กำหนดให้ขา 6 รับสัญญาณ
decode_results results;
void setup()
{ Serial.begin(9600);                         //กำหนดการสื่อสารแบบอนุกรม
  irrecv.enableIRIn();                        //เริ่มการทำงานของรีโมท
}
void loop()
{ if (irrecv.decode(&results))                //ถ้ากดรีโมทให้รับค่าและถอดรหัส
{ Serial.println(results.value,HEX);          //แสดงค่าที่อ่านได้เป็นเลขฐาน 16
  irrecv.resume();                            //รับค่ารีโมทต่อไป
}
delay(100);
}
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงค่าเป็นเลขฐาน 16 เมื่อกดเลข 1 จะแสดง FF6897 กดเลข 2 แสดง FF9867 และเมื่อกดปุ่มซ้ำจะได้ค่า FFFFFFFF แสดงดังรูป



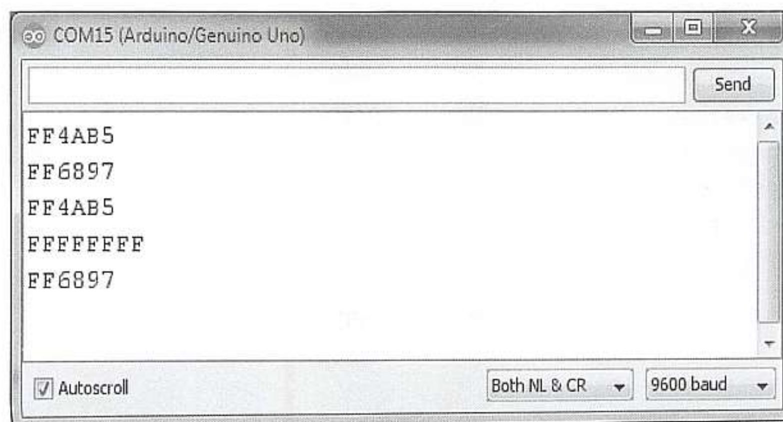
รูปที่ 5.19 ค่าที่ได้จากการกดรีโมท

ตัวอย่างที่ 5.10 โปรแกรมปิด-เปิดไฟที่ขา 13 ด้วยรีโมทอินฟราเรด

```
#include <IRremote.h>                      //เรียกใช้ไลบรารี IRremote.h
int RECV_PIN = 6;
IRrecv irrecv(RECV_PIN);                   //กำหนดขา 6 ให้รับสัญญาณ
decode_results results;
void setup()
{ Serial.begin(9600);
  irrecv.enableIRIn();                     //เริ่มการทำงานของรีโมท
  pinMode(13,OUTPUT);                      //กำหนดให้ขา 13 เป็นเอาต์พุต
}
  void loop()
{ if(irrecv.decode(&results))              //ถ้ากดรีโมทให้รับค่าและถอดรหัส
  { if(results.value == 0xFF6897)          //ถ้ากดเลข 1
    digitalWrite(13,HIGH);                 //ให้ LED ที่ขา 13 ติด
  if(results.value == 0xFF4AB5)            //ถ้ากดเลข 0
    digitalWrite(13,LOW);                  //ให้ LED ที่ขา 13 ดับ
  Serial.println(results.value,HEX);       //แสดงค่าที่รับได้เป็นเลขฐาน 16
  irrecv.resume();                         //รับค่ารีโมทต่อไป
  }
  delay(100);
}
```

ผลการรันโปรแกรม

รีโมทจะควบคุมการปิด-เปิดหลอด LED ที่ขา 13 ตามการกดรีโมทเลข 0 (FF4AB5) และเลข 1 (FF6897) พร้อมแสดงค่าจากการกดรีโมทเป็นเลขฐาน 16 บนจอภาพ และเมื่อกดปุ่มซ้ำจะแสดงค่า FFFFFFFF

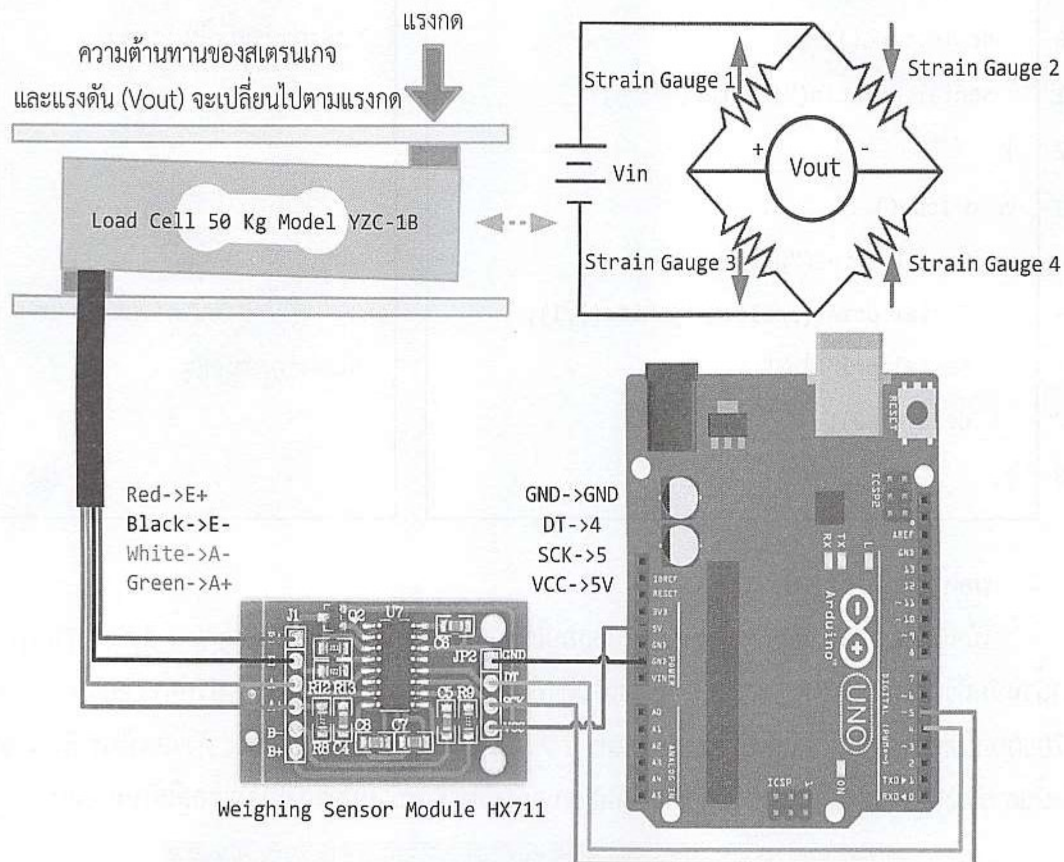


รูปที่ 5.20 ค่าที่ได้จากการกดรีโมทเลข 0 และ 1

5.8 โหลดเซลล์

โหลดเซลล์ (Load Cell) คืออุปกรณ์ที่ใช้ตรวจวัดน้ำหนักสิ่งของต่าง ๆ โดยมีสเตรนเกจ (Strain Gauge) ติดตั้งอยู่ 4 จุด หลักการทำงานของโหลดเซลล์ เมื่อมีแรงกดหรือแรงดึงมากระทำต่อโหลดเซลล์จะทำให้ความต้านทานของสเตรนเกจและสัญญาณไฟฟ้าเปลี่ยนค่าไปตามแรงกด

สเตรนเกจเป็นส่วนประกอบหลักของโหลดเซลล์ ทำจากเส้นลวดโลหะขนาดเล็กขดเป็นรูปร่างต่าง ๆ อยู่บนแผ่นฉนวนหรือสารกึ่งตัวนำเพื่อใช้ในการตรวจวัดแรงของวัตถุ หลักการทำงาน เมื่อสเตรนเกจถูกแรงมากระทำจะทำให้เกิดการเปลี่ยนแปลงรูปร่าง บิด งอ ความต้านทานและแรงดันไฟฟ้าเอาต์พุตจึงเปลี่ยนแปลงตามไปด้วย ทำให้สามารถวัดแรงและน้ำหนักของวัตถุได้ ในการใช้งานโหลดเซลล์จะเชื่อมต่อผ่านบอร์ด HX711 ซึ่งทำหน้าที่ขยายและแปลงสัญญาณเพื่อส่งให้ไมโครคอนโทรลเลอร์ แสดงดังรูปที่ 5.21



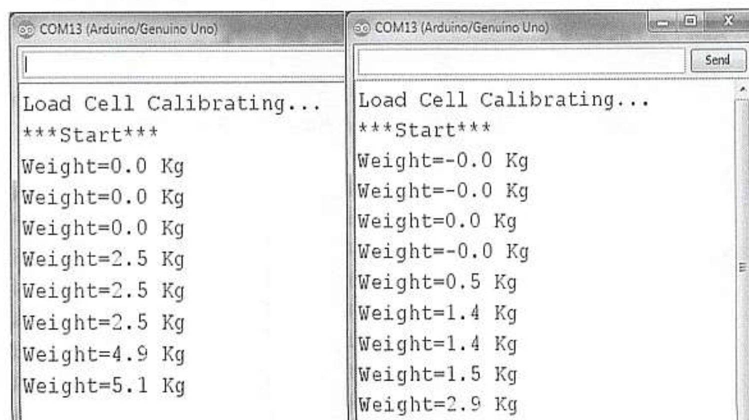
รูปที่ 5.21 วงจรการเชื่อมต่อ Arduino กับโหลดเซลล์

ตัวอย่างที่ 5.11 โปรแกรมอ่านค่าจากโหลดเซลล์

```
#include <HX711.h>                              //เรียกใช้ไลบรารี HX711.h
#define calibration 87000                       //กำหนดการปรับค่าน้ำหนักที่วัดได้
#define DT 4                                    //กำหนดค่า DT = 4
#define SCK 5                                   //กำหนดค่า SCK = 5
HX711 scale(DT,SCK);                            //ขา 4 ต่อ DT ขา 5 ต่อ SCK
void setup()
{ Serial.begin(9600);                           //กำหนดอัตราการสื่อสารข้อมูล
  Serial.println("Load Cell Calibrating...");
  scale.set_scale(calibration);                 //ปรับค่าน้ำหนักที่วัดได้
  scale.tare();                                 //รีเซ็ตค่าน้ำหนักให้เป็นศูนย์
  Serial.println("***Start***");                //แสดงข้อความ
}
void loop()
  { Serial.print("Weight= ");                   //แสดงข้อความ
    Serial.print(scale.get_units(),1);          //แสดงค่าน้ำหนักเป็นทศนิยม 1 ตำแหน่ง
    Serial.println(" Kg");                      //แสดงข้อความ Kg
    delay(200);
}
```

ผลการรันโปรแกรม

เมื่อปรับค่า calibration = 50000 ทดสอบโดยการใช้ลูกตุ้มน้ำหนัก 1.5 Kg และ 3 Kg ปรากฏว่าค่าน้ำหนักที่วัดได้เป็น 2.5 Kg และ 5.1 Kg ซึ่งเป็นค่าที่มีความผิดพลาดมาก ดังนั้นจึงปรับค่า calibration = 87000 จะได้ค่าน้ำหนักที่วัดได้เป็น 1.4 Kg และ 2.9 Kg ซึ่งใกล้เคียงกับน้ำหนักจริง ดังแสดงในรูปที่ 5.22 ดังนั้นการใช้งานโหลดเซลล์ต้องปรับค่า calibration และเลือกช่วงน้ำหนักของโหลดเซลล์ให้เหมาะสม



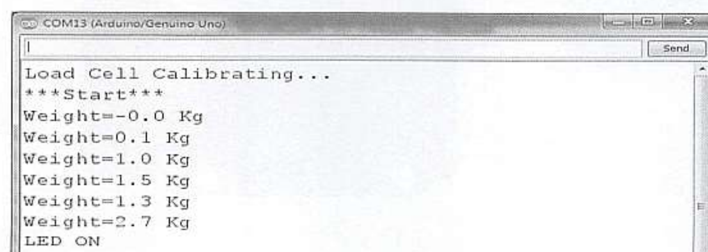
รูปที่ 5.22 ผลการแสดงน้ำหนักเมื่อปรับค่า calibration เป็น 50000 และ 87000

ตัวอย่างที่ 5.12 โปรแกรมแจ้งเตือนเมื่อน้ำหนักเกินค่าที่กำหนด

```
#include <HX711.h>                                //เรียกใช้ไลบรารี HX711.h
#define calibration 87000                         //กำหนดการปรับค่าน้ำหนักที่วัดได้
HX711 scale(4,5);                                 //ขา 4 ต่อ DT ขา 5 ต่อ SCK
void setup()
{ Serial.begin(9600);                             //กำหนดอัตราการสื่อสารข้อมูล
  pinMode(13,OUTPUT);
  Serial.println("Load Cell Calibrating...");
  scale.set_scale(calibration);                   //ปรับค่าน้ำหนักที่วัดได้
  scale.tare();                                   //รีเซ็ตค่าน้ำหนักให้เป็นศูนย์
  Serial.println("***Start***");                  //แสดงข้อความ
}
void loop()
{ Serial.print("Weight= ");                       //แสดงข้อความ
  Serial.print(scale.get_units(),1);              //แสดงค่าน้ำหนักเป็นทศนิยม 1 ตำแหน่ง
  Serial.println(" Kg");                          //แสดงข้อความ Kg
  if(scale.get_units()>2)                         //ถ้าน้ำหนักเกิน 2 Kg
    { digitalWrite(13,HIGH);                      //ให้ขา 13 เป็น HIGH
      Serial.println("LED ON");                   //แสดงข้อความ LED ON
    }
  else                                            //น้ำหนักไม่เกิน 2 Kg
  digitalWrite(13,LOW);                           //ให้ขา 13 เป็น LOW
delay(200);
}
```

ผลการรันโปรแกรม

เมื่อน้ำหนักที่วัดได้มีค่ามากกว่า 2 Kg จะทำให้ขา 13 เป็น HIGH พร้อมกับแสดงข้อความ LED ON



รูปที่ 5.23 ผลการแสดงน้ำหนักและแจ้งเตือนเมื่อน้ำหนักเกินค่าที่ตั้งไว้

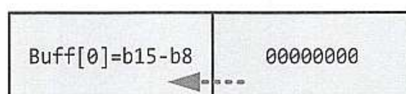
5.9 เซนเซอร์แสง

เซนเซอร์แสง BH1750 เป็นเซนเซอร์ที่ใช้วัดความเข้มของแสง มีความละเอียด 16 บิต หน่วยเป็นลักซ์ (Lux) ใช้การเชื่อมต่อกับไมโครคอนโทรลเลอร์แบบ I2C (SDA SCL) การอ่านค่าแสงจากเซนเซอร์ BH1750 จะอ่านค่าที่ตำแหน่ง 0x23 จำนวน 2 ไบต์ โดยไบต์แรกเก็บไว้ใน Buff[0] แล้วเลื่อน 8 บิตนำมาบวกกับไบต์ที่สองที่เก็บไว้ใน Buff[1] แล้วจะได้ค่าความเข้มของแสง การอ่านค่าแสงและวงจรการเชื่อมต่อแสดงดังรูปที่ 5.24 สามารถเขียนโปรแกรมได้ดังตัวอย่างที่ 5.13

อ่านค่าแสงไบต์แรก Buff[0] = Wire.read(); | Buff[0]=b7-b0

อ่านค่าแสงไบต์ที่สอง Buff[1] = Wire.read(); | Buff[1]=b7-b0

Light=((Buff[0]<<8)|Buff[1]);

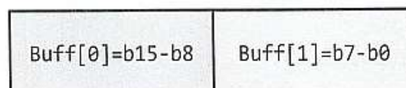


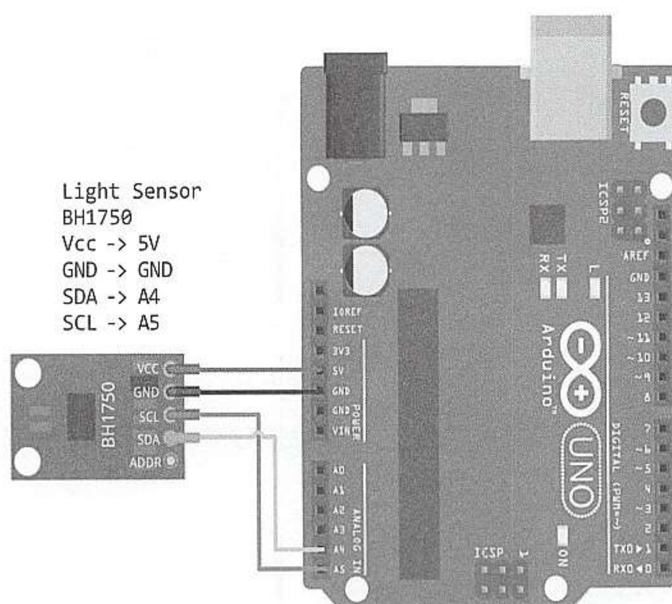
เลื่อน 8 บิต

OR

Buff[1]=b7-b0

ไบต์แรก OR ไบต์ที่สอง





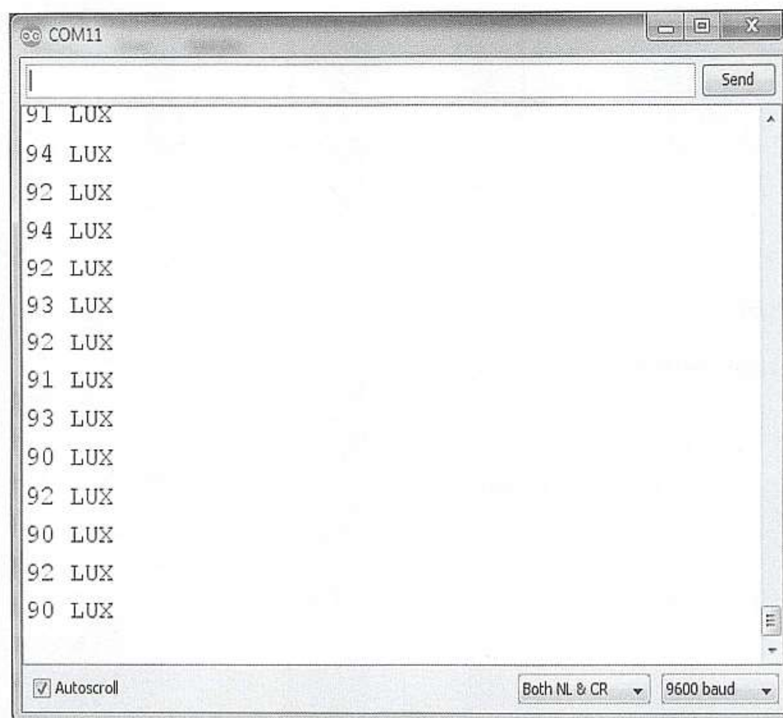
รูปที่ 5.24 การเชื่อมต่อ Arduino กับเซนเซอร์แสง

ตัวอย่างที่ 5.13 อ่านค่าแสงจากเซนเซอร์ BH1750

```
#include <Wire.h>                              //เรียกใช้ไลบรารี Wire.h
int AddBH1750 = 0x23;                          //กำหนดแอดเดรสของเซนเซอร์
byte Buff[2];                                  //ประกาศตัวแปรอาร์เรย์
void setup()
{ Serial.begin(9600);
  Wire.begin();                                //เริ่มการสื่อสาร I2C
  Wire.beginTransmission(AddBH1750);           //ติดต่อเซนเซอร์ BH1750
  Wire.write(0x10);                            //เริ่มการใช้งาน
  Wire.endTransmission();                      //จบการสื่อสาร
}
void loop()
{ int Light = 0;
  Wire.beginTransmission(AddBH1750);           //ติดต่อเซนเซอร์ BH1750
  Wire.requestFrom(AddBH1750, 2);              //ร้องขอค่าข้อมูลจำนวน 2 ไบต์
  Buff[0] = Wire.read();                       //อ่านค่าแสงไบต์แรก
  Buff[1] = Wire.read();                       //อ่านค่าแสงไบต์ที่สอง
  Wire.endTransmission();
  Light=((Buff[0]<<8)|Buff[1]);                //เลื่อนไบต์แรกแล้วรวมค่าทั้งสองไบต์
  Serial.print(Light);                         //แสดงค่าความเข้มของแสง
  Serial.println(“ LUX”);
  delay(200);
}
```

ผลการรันโปรแกรม

โปรแกรมอ่านค่าแสงจากเซนเซอร์ BH1750 ที่ตำแหน่ง 0x23 จำนวน 2 ไบต์ โดยไบต์แรกเก็บไว้ใน Buff[0] แล้วเลื่อนข้อมูล 8 บิตแรกนำมาบวกกับไบต์ที่สองที่เก็บไว้ใน Buff[1] ก็จะได้ค่าแสง (Light=Buff[0](b15–b8)+Buff[1](b7–b0)) แสดงผลการรันโปรแกรมดังรูป



รูปที่ 5.25 ผลการวัดค่าแสง

5.10 สรุป

เซนเซอร์ คืออุปกรณ์ตรวจจับสัญญาณทางฟิสิกส์ เช่น อุณหภูมิ ความชื้น เสียง แสง การสัมผัส หรือการเคลื่อนไหว

เซนเซอร์แบบสัมผัส ใช้แทนการทำงานของสวิตช์ปกติ โดยจะทำงานเมื่อมีการสัมผัส

PIR Motion Sensor คือเซนเซอร์ตรวจจับความเคลื่อนไหวด้วยคลื่นรังสีความร้อนหรือคลื่นอินฟราเรด เมื่อสิ่งมีชีวิตเคลื่อนไหว เซนเซอร์สามารถตรวจจับคลื่นความร้อนที่เปลี่ยนแปลงแล้วส่งสัญญาณเอาต์พุตแบบดิจิทัลออกมา

เซนเซอร์วัดแรงดันไฟฟ้า สามารถวัดแรงดันไฟฟ้ากระแสตรงได้ 0–24 โวลต์ ตัวเซนเซอร์จะมีขาสัญญาณ 3 ขา คือ ขากราวด์ ขาสัญญาณไฟ และขาสัญญาณเอาต์พุตแอนะล็อก

เซนเซอร์วัดระยะทางแบบอัลตราโซนิก HC-SR04 เป็นเซนเซอร์ที่ใช้คลื่นเสียงสะท้อนในการวัดระยะ โดยสามารถวัดระยะทางได้ในช่วง 2–400 เซนติเมตร

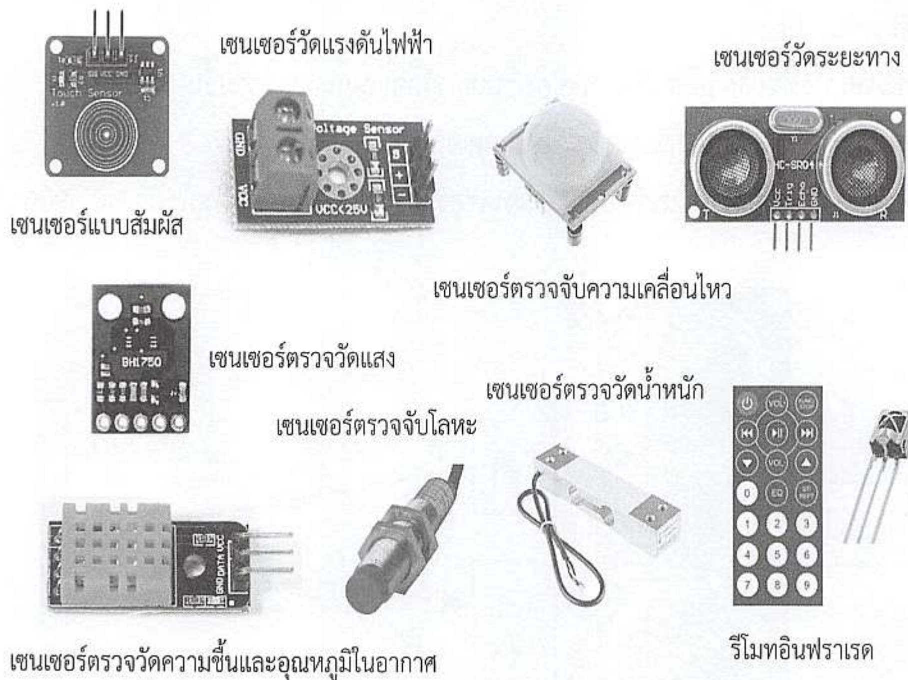
DHT (Digital Humidity and Temperature Sensor) เป็นเซนเซอร์ตรวจวัดความชื้นและอุณหภูมิในอากาศ สามารถวัดความชื้นสัมพัทธ์ 20–90% และวัดอุณหภูมิได้ในช่วง 0–50 องศาเซลเซียส

เซนเซอร์ตรวจจับโลหะหรือพร็อกซิมิตี้เซนเซอร์แบบอินดักทีฟ ใช้ในการตรวจจับโลหะ เช่น เหล็ก อะลูมิเนียม มีระยะตรวจจับสูงสุด 4 มิลลิเมตร

รีโมทอินฟราเรด คือตัวควบคุมระยะไกลโดยใช้คลื่นอินฟราเรดในการรับและส่งสัญญาณ โดยมีตัวส่งสัญญาณคือรีโมท และมีตัวรับสัญญาณอินฟราเรดซึ่งจะต่อเข้าบอร์ด Arduino

โหลดเซลล์ คืออุปกรณ์ที่ใช้ตรวจวัดน้ำหนักสิ่งของ เมื่อมีแรงกดจะทำให้ความต้านทานของสเตรนเกจและสัญญาณไฟฟ้าเปลี่ยนค่าไปตามแรงที่กระทำต่อโหลดเซลล์

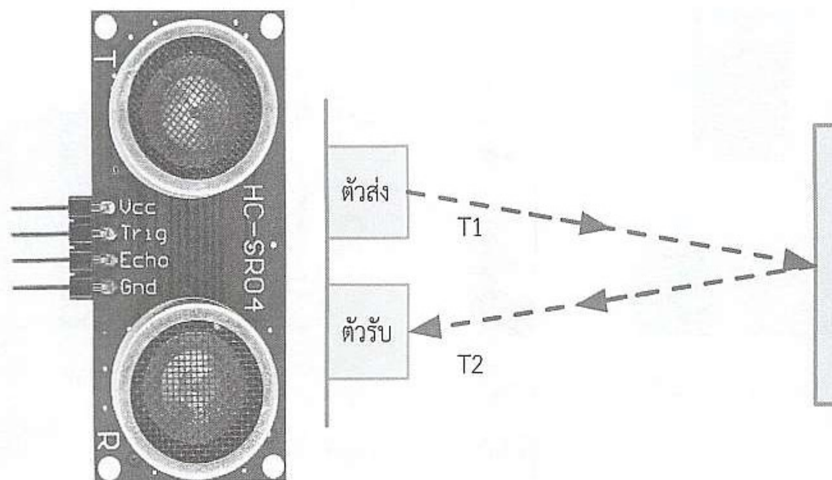
BH1750 เป็นเซนเซอร์ที่ใช้วัดความเข้มของแสง มีความละเอียด 16 บิต มีหน่วยเป็นลักซ์



รูปที่ 5.26 เซนเซอร์แบบต่าง ๆ

แบบฝึกหัดบทที่ 5

1. อธิบายหลักการทำงานและขาสัญญาณของ PIR Motion Sensor
2. คลื่นอินฟราเรดคืออะไร
3. อธิบายหลักการทำงานและขาสัญญาณของเซนเซอร์วัดอุณหภูมิและความชื้น
4. อธิบายหลักการทำงานและขาสัญญาณของโหลดเซลล์
5. อธิบายขาสัญญาณและหลักการทำงานของเซนเซอร์วัดระยะทางแบบอัลตราโซนิก HC-SR04



รูปที่ 5.27 เซนเซอร์ HC-SR04

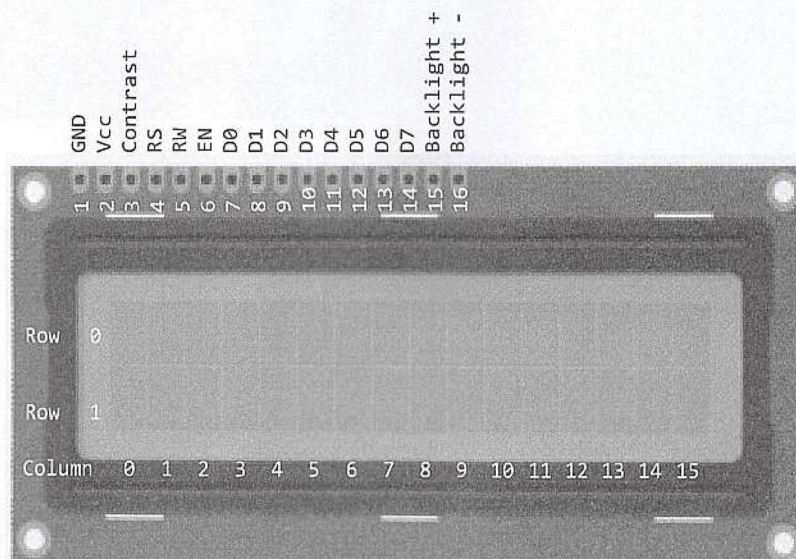
6. ให้เขียนโปรแกรมวัดระยะทางในหน่วยมิลลิเมตร
7. ให้เขียนโปรแกรมและวงจรควบคุมอุณหภูมิ หากอุณหภูมิเกิน 35 องศาเซลเซียสให้พัดลมทำงาน
8. ให้เขียนโปรแกรมและวงจรควบคุมการปิด-เปิดประตูอัตโนมัติเมื่อมีคนเดินผ่าน

บทที่
6 จอแสดงผล

จอแสดงผลแบบ LCD (Liquid Crystal Display) สามารถแสดงผลได้ทั้งตัวเลข ตัวอักษร รูปภาพ สัญลักษณ์ และกราฟิก ปัจจุบันด้วยราคาที่ถูกลงมากจึงทำให้ได้รับความนิยมใช้งานอย่างแพร่หลายในการแสดงผลข้อมูลหรือแสดงสถานะการทำงานของระบบควบคุม ซึ่งในบทนี้จะกล่าวถึงการใช้งานจอแสดงผล LCD และจอ LCD แบบ I2C จอแสดงผล OLED จอแสดงผล 7 ส่วน รวมทั้งการเขียนโปรแกรมควบคุมการแสดงผลในรูปแบบต่าง ๆ

6.1 จอแสดงผล LCD

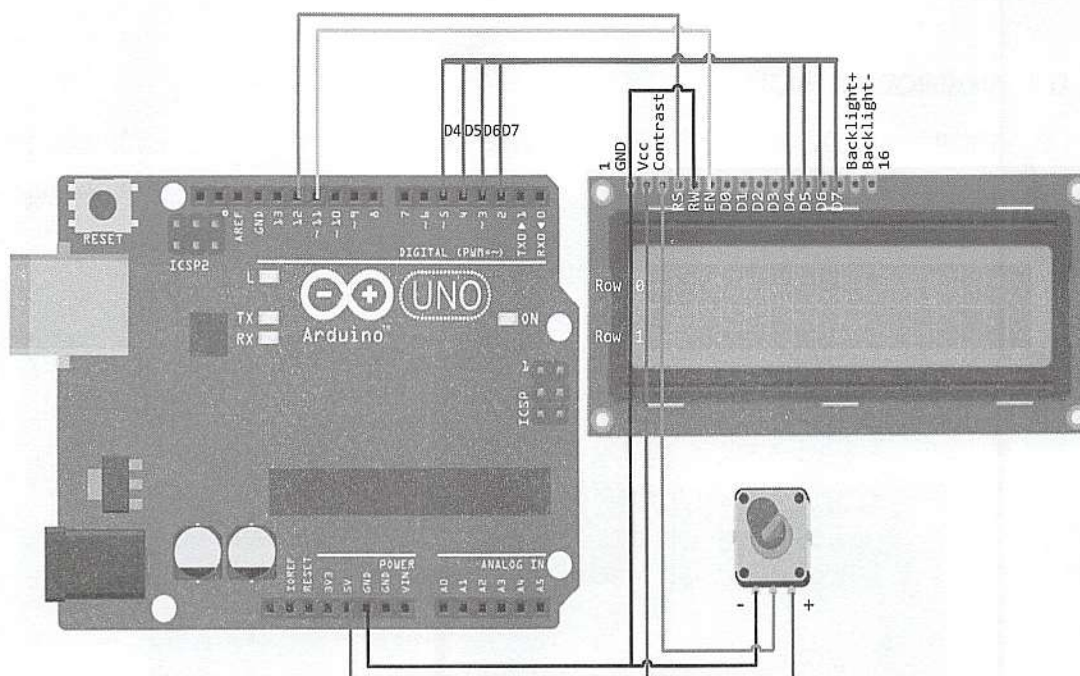
จอแสดงผล LCD มีอยู่หลายชนิด มีทั้งจอแสดงผล LCD แบบ 16 ตัวอักษร 2 บรรทัด และ 20 ตัวอักษร 4 บรรทัด ความละเอียดในการแสดงผล 5x7 จุด มีขาสัญญาณข้อมูลขนาด 8 บิต ลักษณะของจอแสดงผล LCD และขาสัญญาณ แสดงดังรูปที่ 6.1 และตารางที่ 6.1



รูปที่ 6.1 ขาสัญญาณจอแสดงผล LCD ขนาด 16x2

ตารางที่ 6.1 หน้าที่การทำงานขาสัญญาณของจอแสดงผล LCD

ตำแหน่งขา	หน้าที่
1	GND หรือ Vss เป็นขากราวด์ของจอ LCD
2	Vcc เป็นขาสัญญาณไฟฟ้ากระแสตรง 5 โวลต์
3	Contrast เป็นขาสัญญาณไฟที่ใช้ปรับความคมชัดในการแสดงผลของจอ LCD
4	RS เป็นขาสัญญาณที่ใช้ควบคุม ถ้าขา RS = 0 ข้อมูลที่ส่งมาทางขา D0–D7 จะเป็นคำสั่ง แต่ถ้าขา RS = 1 จะเป็นข้อมูลที่ส่งออกมาแสดงผล
5	RW เป็นขาสัญญาณควบคุมการอ่านและเขียน ถ้าขา RW = 0 แสดงว่าต้องการเขียนข้อมูลออกจอ LCD แต่ถ้า RW = 1 แสดงว่าต้องการอ่านข้อมูลจากจอ LCD
6	EN เป็นขาสัญญาณ Enable ที่จะต้องส่งสัญญาณพัลส์ 1 ลูกเมื่อมีการส่งข้อมูลหรือคำสั่งมายังจอแสดงผล LCD
7–14	D0–D7 เป็นขาสัญญาณข้อมูล
15	A เป็นขาสัญญาณไฟกระแสตรง 5 โวลต์ เมื่อต้องการให้จอ LCD สว่าง
16	K เป็นขาสัญญาณกราวด์เมื่อต้องการให้จอ LCD สว่าง



รูปที่ 6.2 วงจรการเชื่อมต่อ Arduino กับจอแสดงผล LCD

จากรูปที่ 6.2 ขาสัญญาณ RS ต่อกับขา 12 ขา EN ต่อกับขา 11 และขา D4 D5 D6 D7 ของจอ LCD ต่อกับขา 5 4 3 2 ของบอร์ด Arduino ซึ่งมีการรับและส่งข้อมูลแบบ 4 บิต และมีความต้านทานแบบปรับค่าได้ทำหน้าที่ปรับความสว่างของจอภาพ

6.2 ฟังก์ชันควบคุมจอแสดงผล LCD

ในการเขียนโปรแกรมควบคุมการทำงานของจอแสดงผล LCD จำเป็นต้องเรียกใช้ไลบรารี #include <LiquidCrystal.h> เพื่อให้สามารถเรียกใช้งานฟังก์ชันต่าง ๆ ในการควบคุมจอ LCD ได้สะดวก โดยมีฟังก์ชันที่จำเป็นต่อการใช้งานดังตัวอย่างต่อไปนี้

ตัวอย่าง

```
LiquidCrystal lcd(12,11,5,4,3,2);
```

หมายถึง กำหนดขาสัญญาณ RS ต่อกับขา 12 ขา EN ต่อกับขา 11 ขา D4 D5 D6 D7 ต่อกับขา 5 4 3 2 ของบอร์ด

```
lcd.begin(16,2);
```

หมายถึง กำหนดขนาดจอแสดงผล LCD ให้มีขนาด 16 ตัวอักษร 2 บรรทัด

```
lcd.print("Arduino by Adon");
```

หมายถึง แสดงข้อความ Arduino by Adon ออกจอแสดงผล LCD

```
int a=100;
lcd.print(a);
```

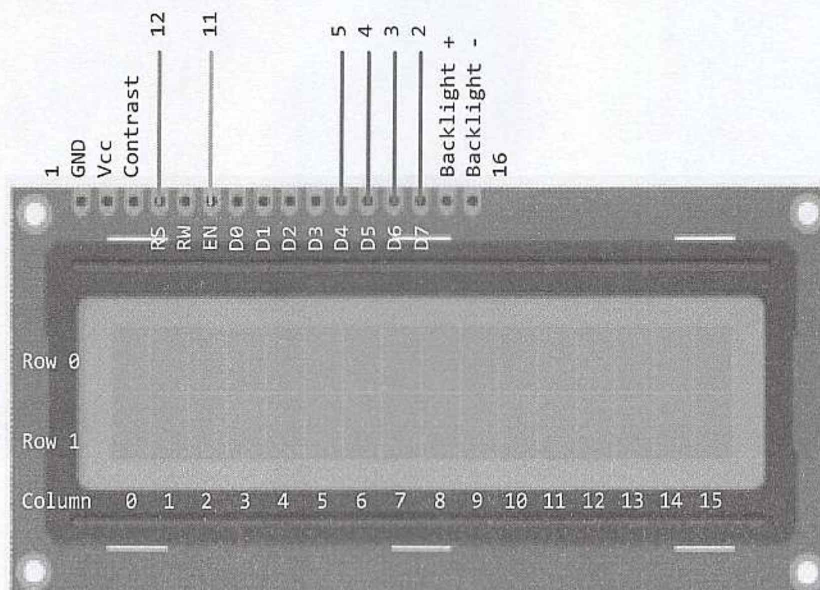
หมายถึง แสดงตัวเลข 100 ออกจอแสดงผล LCD

```
lcd.setCursor(2,1);
```

หมายถึง กำหนดตำแหน่งในการแสดงผลที่ตำแหน่ง 2,1 (แสดงผลในคอลัมน์ที่ 2 (Column 2) และบรรทัดที่ 1 (Row 1) คอลัมน์และบรรทัดมีค่าเริ่มต้นที่ 0)

```
lcd.clear();
```

หมายถึง ลบหน้าจอแสดงผล LCD



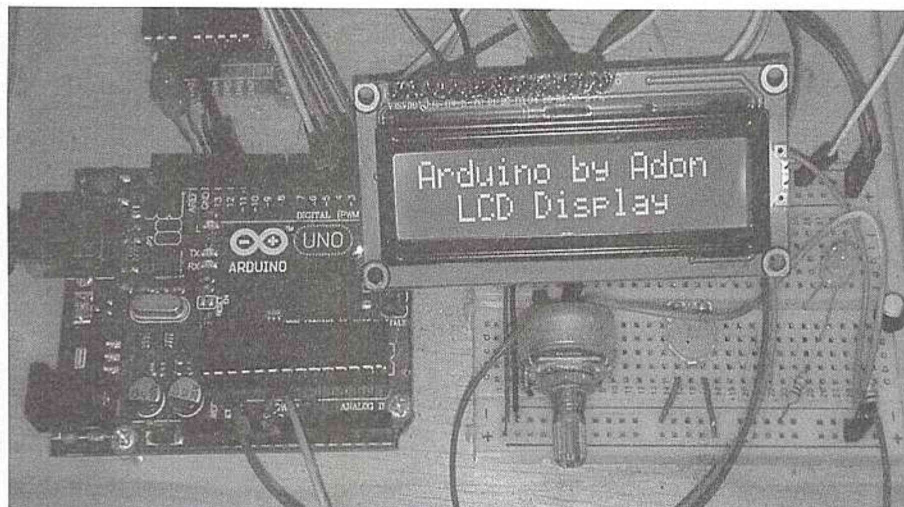
รูปที่ 6.3 การต่อขาสัญญาณ RS EN D4 D5 D6 และ D7

ตัวอย่างที่ 6.1 โปรแกรมแสดงข้อความออกจอแสดงผล LCD

```
#include <LiquidCrystal.h>            //เรียกใช้ไลบรารี LiquidCrystal.h
LiquidCrystal lcd(12,11,5,4,3,2);     //กำหนดขาสัญญาณในการเชื่อมต่อ
void setup()
{
  lcd.begin(16,2);                    //กำหนด LCD 16 ตัวอักษร 2 บรรทัด
  lcd.print("Arduino by Adon");       //แสดงข้อความ Arduino by Adon
}
void loop()
{
  lcd.setCursor(2,1);                 //แสดงผลในคอลัมน์ที่ 2 บรรทัดที่ 1
  lcd.print("LCD Display");           //แสดงข้อความ LCD Display
}
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงข้อความ Arduino by Adon ในบรรทัดที่ 0 และแสดงข้อความ LCD Display เริ่มในตำแหน่งที่ 2 บรรทัดที่ 1 หากต้องการให้จอสว่าง ต้องต่อไฟเข้าขา 15 และต่อกราวด์เข้ากับขา 16 ของจอแสดงผล LCD



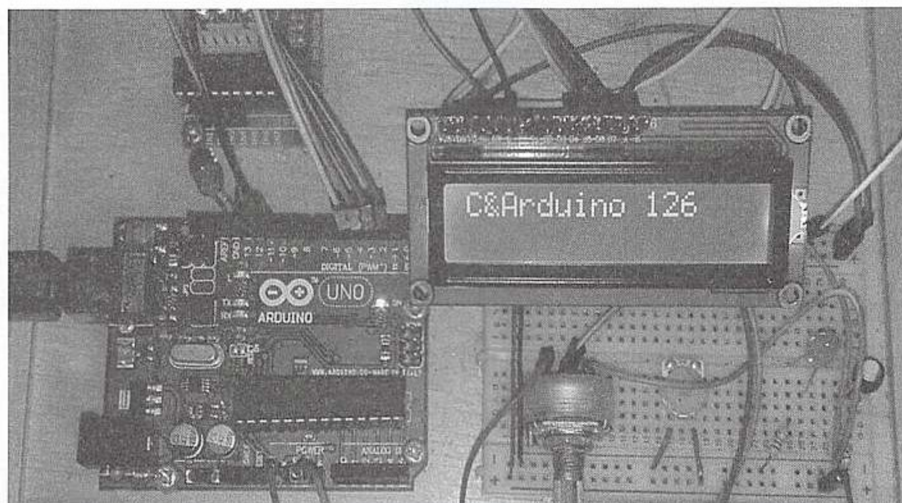
รูปที่ 6.4 การแสดงข้อความบนจอแสดงผล LCD

ตัวอย่างที่ 6.2 โปรแกรมแสดงข้อความและเวลา

```
#include <LiquidCrystal.h>            //เรียกใช้ไลบรารี LiquidCrystal.h
LiquidCrystal lcd(12,11,5,4,3,2);
void setup()
{
  lcd.begin(16,2);                    //กำหนด LCD 16 ตัวอักษร 2 บรรทัด
  lcd.print("C&Arduino");             //แสดงข้อความ C&Arduino
}
void loop()
{
  lcd.setCursor(10,0);                //แสดงผลในคอลัมน์ที่ 10 บรรทัดที่ 0
  lcd.print(millis()/1000);           //แสดงเวลาของ millis()
}
```

ผลการรันโปรแกรม

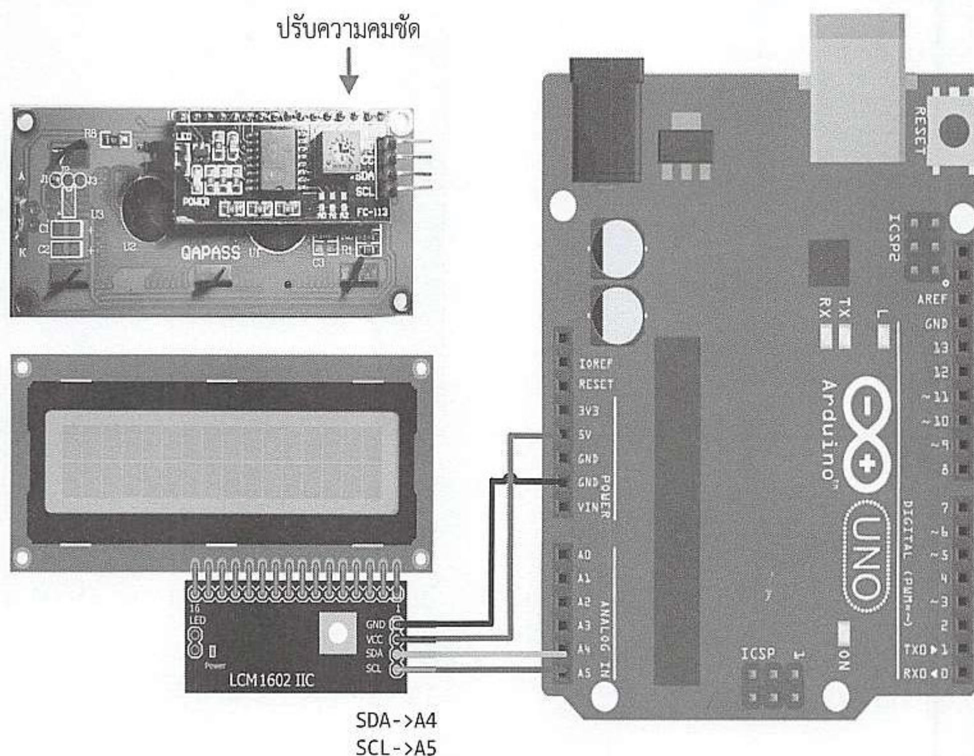
โปรแกรมจะแสดงข้อความ C&Arduino และเวลาในบรรทัดที่ 0 โดยฟังก์ชัน millis()/1000; เป็นการแสดงเวลาในหน่วยวินาที ซึ่งจะเริ่มนับอัตโนมัติเมื่อเริ่มรันโปรแกรม



รูปที่ 6.5 การแสดงข้อความและเวลาบนจอแสดงผล LCD

6.3 จอแสดงผล LCD แบบ I2C

จอแสดงผล LCD แบบ I2C มีข้อดีคือใช้สายสัญญาณควบคุม 2 เส้นทำให้สะดวกในการเชื่อมต่อกับไมโครคอนโทรลเลอร์ การเขียนโปรแกรมต้องเรียกใช้ไลบรารี #include <LiquidCrystal_I2C.h> จึงจะสามารถเรียกใช้งานฟังก์ชันต่าง ๆ ในการควบคุมการแสดงผลได้ การเชื่อมต่อจอแสดงผล LCD แบบ I2C ขนาด 16x2 และ LCD ขนาด 20x4 จะต่อแบบเดียวกับการต่อจอแสดงผล LCD ขนาด 16x2 ดังแสดงในรูปที่ 6.6



รูปที่ 6.6 การเชื่อมต่อ Arduino กับจอแสดงผล LCD แบบ I2C

I2C (Inter-Integrated Circuit) ถูกพัฒนาขึ้นโดยบริษัทฟิลิปส์ (Philips) เป็นการสื่อสารข้อมูลอนุกรมแบบซิงโครนัส (Synchronous) เพื่อใช้ติดต่อสื่อสารระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์อินพุตและเอาต์พุต โดยใช้สายสัญญาณเพียง 2 เส้น คือสายสัญญาณข้อมูลแบบอนุกรม (Serial Data Line : SDA) และสายสัญญาณนาฬิกา (Serial Clock Line : SCL) ซึ่งมีข้อดีคือสามารถเชื่อมต่ออุปกรณ์หลายตัวเข้าด้วยกันได้โดยใช้สายสัญญาณเพียง 2 เส้น

6.4 ฟังก์ชันควบคุมจอแสดงผล LCD แบบ I2C

การเขียนโปรแกรมควบคุมจอแสดงผล LCD แบบ I2C ต้องเรียกใช้ไลบรารี #include <LiquidCrystal_I2C.h> จึงสามารถเรียกใช้ฟังก์ชันควบคุมการแสดงผลได้ ซึ่งมีฟังก์ชันพื้นฐานที่ใช้งานดังนี้

ฟังก์ชัน LiquidCrystal_I2C lcd(Address,columns,rows); กำหนดตำแหน่งและขนาดของจอแสดงผล

ฟังก์ชัน lcd.begin(); กำหนดค่าเริ่มต้นให้จอแสดงผลทำงาน

ฟังก์ชัน lcd.clear(); ลบจอภาพทั้งหมด

ฟังก์ชัน lcd.setCursor(); กำหนดตำแหน่งของการแสดงผล

ฟังก์ชัน lcd.print(); ใช้แสดงข้อความ

ตัวอย่าง

```
LiquidCrystal_I2C lcd(0x27,16,2);
LiquidCrystal_I2C lcd(0x3F,16,2);
```

หมายถึง ตำแหน่งหรือที่อยู่ (Address) ของจอแสดงผลคือ 27 (ค่าตำแหน่งปกติจะเป็น 27 หรือ 3F) ขนาดของจอ 16 ตัวอักษร 2 บรรทัด

```
LiquidCrystal_I2C lcd(0x27,20,4);
LiquidCrystal_I2C lcd(0x3F,20,4);
```

หมายถึง ตำแหน่งหรือที่อยู่ของจอแสดงผลคือ 27 หรือ 3F ขนาดของจอ 20 ตัวอักษร 4 บรรทัด

```
lcd.setCursor(0,1);
lcd.print("Adonson");
```

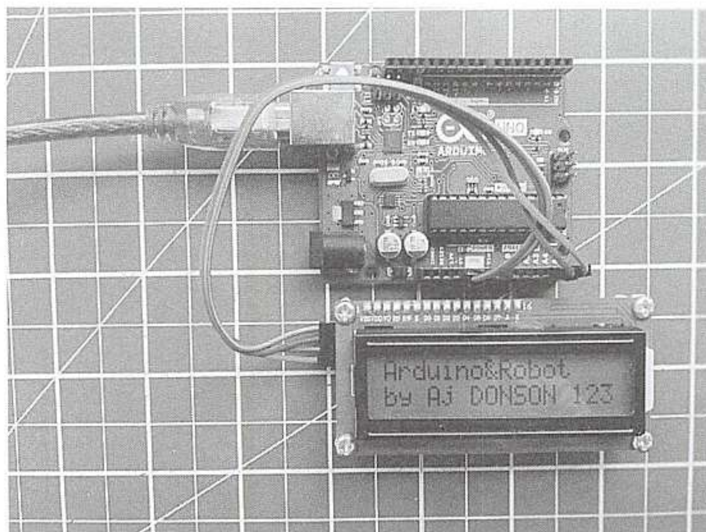
หมายถึง แสดงข้อความ Adonson ในคอลัมน์ที่ 0 บรรทัดที่ 1 (การนับคอลัมน์และบรรทัดเริ่มจาก 0)

ตัวอย่างที่ 6.3 โปรแกรมแสดงข้อความบนจอ LCD แบบ I2C ขนาด 16x2

```
#include <LiquidCrystal_I2C.h>              //เรียกใช้ไลบรารี LiquidCrystal_I2C.h
LiquidCrystal_I2C lcd(0x27,16,2);           //กำหนดแอดเดรสและขนาดของจอ
void setup()
{
  //A4->SDA A5->SCL
  lcd.begin();                              //เริ่มการทำงานของจอ LCD
  lcd.print("Arduino&Robot");               //แสดงข้อความ
}
void loop()
{
  lcd.setCursor(0,1);                       //กำหนดตำแหน่งการแสดงผล
  lcd.print("by Aj DONSON 123");            //แสดงข้อความ
  delay(1000);
}
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงข้อความ Arduino&Robot ในบรรทัดที่ 0 และแสดงข้อความ by Aj DONSON 123 ในบรรทัดที่ 1 และสามารถปรับความคมชัดของจอที่ความต้านทานแบบปรับค่าได้บริเวณด้านหลังของจอแสดงผล



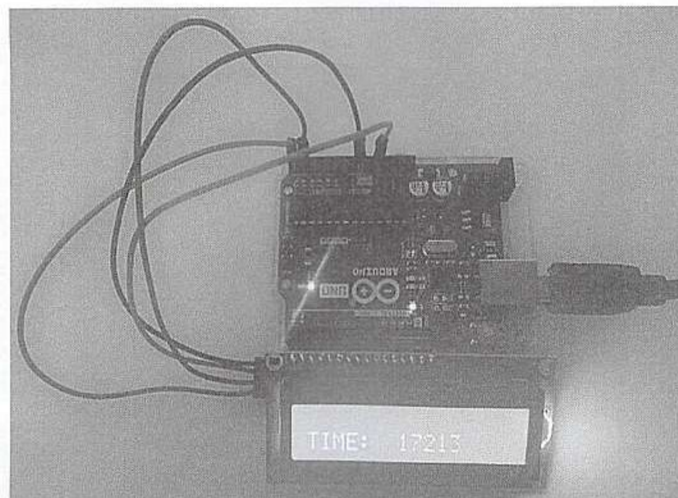
รูปที่ 6.7 การแสดงข้อความบนจอ LCD แบบ I2C

ตัวอย่างที่ 6.4 โปรแกรมแสดงการนับของฟังก์ชัน millis(); บนจอ LCD แบบ I2C ขนาด 16x2

```
#include <LiquidCrystal_I2C.h>          //เรียกใช้ไลบรารี LiquidCrystal_I2C.h
LiquidCrystal_I2C lcd(0x27,16,2);       //กำหนดแอดเดรสและขนาดของจอ
void setup()
{
  //A4->SDA A5->SCL
  lcd.begin();                          //เริ่มการทำงานของจอ LCD
  lcd.setCursor(0,1);                   //กำหนดตำแหน่งการแสดงผล
  lcd.print("TIME:");                   //แสดงข้อความ
}
void loop()
{
  lcd.setCursor(7,1);                   //กำหนดตำแหน่งการแสดงผล
  lcd.print(millis());                  //แสดงเวลาของฟังก์ชัน millis();
  delay(1000);                          //หน่วงเวลา
}
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงข้อความ TIME: และเวลาของฟังก์ชัน millis(); หากจอไม่สามารถแสดงผลได้ให้ปรับแก้ตำแหน่ง 27 ให้เป็น 3F



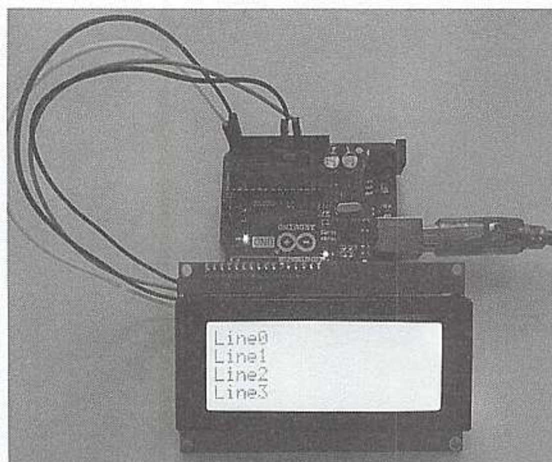
รูปที่ 6.8 การแสดงเวลาของฟังก์ชัน millis(); บนจอ LCD แบบ I2C

ตัวอย่างที่ 6.5 โปรแกรมแสดงข้อความบนจอ LCD แบบ I2C ขนาด 20x4

```
#include <LiquidCrystal_I2C.h>          //เรียกใช้ไลบรารี LiquidCrystal_I2C.h
LiquidCrystal_I2C lcd(0x27,20,4);       //กำหนดแอดเดรสและขนาดของจอ
void setup()
{
  //A4->SDA A5->SCL
  lcd.begin();                          //เริ่มการทำงานของจอ LCD
}
void loop()
{
  lcd.setCursor(0,0);                   //กำหนดตำแหน่งการแสดงผล
  lcd.print("Line0");                   //แสดงข้อความ Line0
  lcd.setCursor(0,1);
  lcd.print("Line1");                   //แสดงข้อความ Line1
  lcd.setCursor(0,2);
  lcd.print("Line2");                   //แสดงข้อความ Line2
  lcd.setCursor(0,3);
  lcd.print("Line3");                   //แสดงข้อความ Line3
  delay(1000);
 }
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงข้อความ Line0 Line1 Line2 Line3 ในบรรทัดที่ 0 ถึง 3



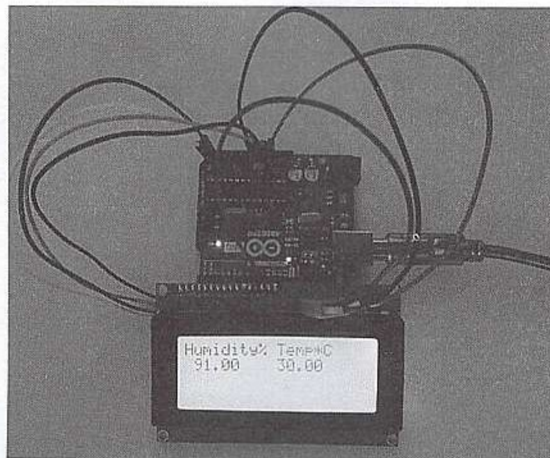
รูปที่ 6.9 การแสดงข้อความบนจอ LCD แบบ I2C ขนาด 20x4

ตัวอย่างที่ 6.6 โปรแกรมแสดงอุณหภูมิและความชื้นบนจอ LCD แบบ I2C ขนาด 20x4

```
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
DHT dht(9,DHT11);
LiquidCrystal_I2C lcd(0x3F,20,4);
void setup()
{ //A4->SDA A5->SCL
  lcd.begin();
  dht.begin();
  lcd.print("Humidity% Temp*C");
}
void loop()
{ delay(1000);
  float h=dht.readHumidity();
  float t=dht.readTemperature();
  lcd.setCursor(1,1);
  lcd.print(h);
  lcd.setCursor(10,1);
  lcd.print(t);
  delay(1000);
}
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงอุณหภูมิและความชื้นที่วัดได้จากเซนเซอร์ DHT11



รูปที่ 6.10 การแสดงอุณหภูมิและความชื้นบนจอ LCD แบบ I2C ขนาด 20x4

6.5 จอแสดงผล OLED

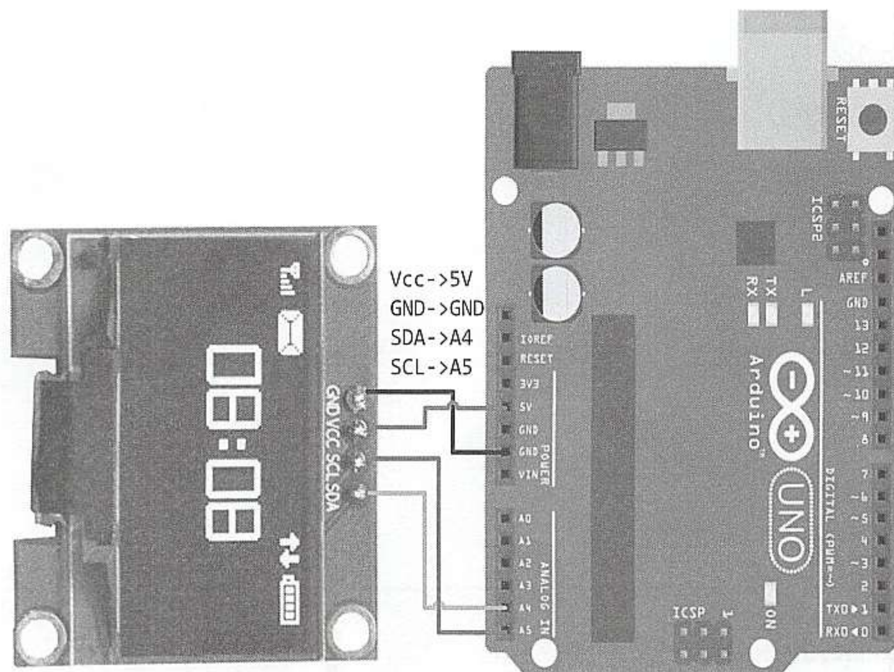
จอแสดงผล OLED (Organic Light-Emitting Diode) คือไดโอดเปล่งแสง (LED) ที่ออกแบบเป็นฟิล์มแผ่นบาง โดยทำปฏิกิริยาคล้ายกับอุปกรณ์เปล่งแสงแบบ LED เม็ดเหลี่ยม แต่มีข้อดีคือสามารถควบคุมตำแหน่งและความสว่างของแสงได้อย่างแม่นยำ จอแสดงผล OLED สามารถแสดงได้ทั้งรูปภาพ ข้อความ และตัวเลขแบบขาวดำได้ โดยมีขาสัญญาณ 4 ขาดังนี้

Vcc คือ ขาสัญญาณไฟต่อกับแรงดันไฟฟ้า 3.3–5 โวลต์

GND คือ ขากราวด์

SDA คือ ขาสัญญาณข้อมูลแบบอนุกรมซึ่งต่อกับขา A4

SCL คือ ขาสัญญาณนาฬิกาซึ่งต่อกับขา A5



รูปที่ 6.11 การเชื่อมต่อ Arduino กับจอ OLED

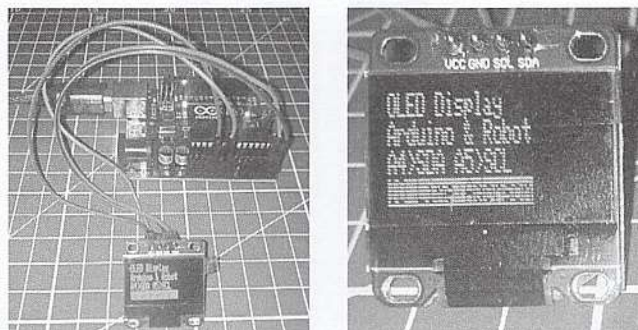
การเชื่อมต่อจะใช้สัญญาณเพียง 2 เส้น คือ SCL ต่อกับขา A5 และ SDA ต่อกับขา A4 ซึ่งเป็นการเชื่อมต่อแบบ I2C ในการเขียนโปรแกรมควบคุมต้องเรียกใช้ไลบรารี #include <Adafruit_GFX.h> และ #include <Adafruit_SSD1306.h>

ตัวอย่างที่ 6.7 โปรแกรมแสดงข้อความและรูปบนจอ OLED

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);
void setup()
{ display.begin(SSD1306_SWITCHCAPVCC,0x3c);  //InitI2C Addr0x3c
  display.clearDisplay();                    //Clears Screen&Buffer
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("OLED Display");
  display.setTextSize(1);
  display.println("Arduino & Robot");
  display.println("A4>SDA A5>SCL");
  display.setTextColor(BLACK,WHITE);
  display.println("ID Line:Adonson");
  display.display();
}
void loop()
  {
  }
```

ผลการรันโปรแกรม

โปรแกรมจะแสดงข้อความบนจอ OLED ดังรูป



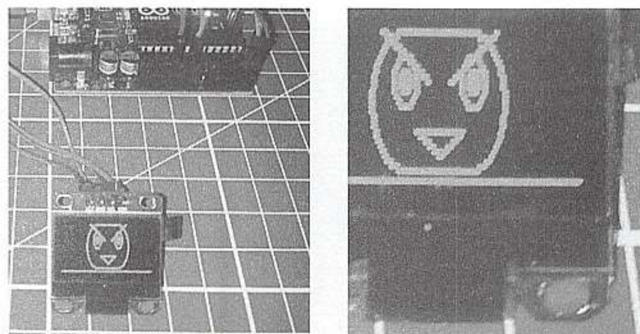
รูปที่ 6.12 การแสดงข้อความบนจอ OLED

ตัวอย่างที่ 6.8 โปรแกรมแสดงกราฟิกบนจอ OLED

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display(4);
void setup()
{ display.begin(SSD1306_SWITCHCAPVCC, 0x3c);//InitI2C Addr0x3c
  display.clearDisplay();                  //Clears Screen&Buffer
  display.drawLine(0,30,127,30, WHITE);
  display.drawLine(70,10,90,0, WHITE);
  display.drawLine(40,0,60,10, WHITE);
  display.drawRoundRect(35,0,60,29,15,WHITE);
  display.drawCircle(50,10,5, WHITE);
  display.fillCircle(50,10,2, WHITE);
  display.drawCircle(80,10,5, WHITE);
  display.fillCircle(80,10,2, WHITE);
  display.drawTriangle(55,20,75,20,65,25, WHITE);
  display.display();
}
void loop()
{
}
```

ผลการรันโปรแกรม

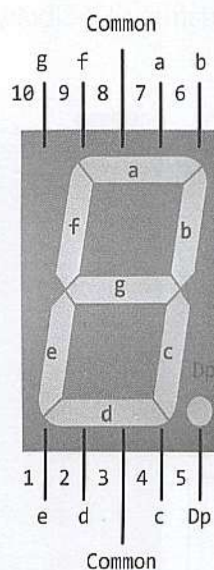
โปรแกรมจะแสดงกราฟิกตามที่กำหนดบนจอ OLED ดังรูป



รูปที่ 6.13 การแสดงกราฟิกบนจอ OLED

6.6 จอแสดงผล 7 ส่วน

จอแสดงผล 7 ส่วน (Seven-Segment Display) คือจอแสดงผลตัวเลข 0–9 และตัวอักษรบางตัวที่สามารถแสดงผลได้ การแสดงผลเกิดจากการจัดวางหลอด LED ในแนวยาวและแนวตั้งซึ่งมีทั้งหมด 7 ส่วน ตามรูปที่ 6.14 คือ ส่วน a b c d e f g และจุด Dp การใช้งานหากต้องการให้แสดงเลขใดต้องทำให้ LED ในส่วนนั้นติด เช่น หาก b และ c ติดก็จะแสดงเลข 1 ส่วน Dp คือจุด



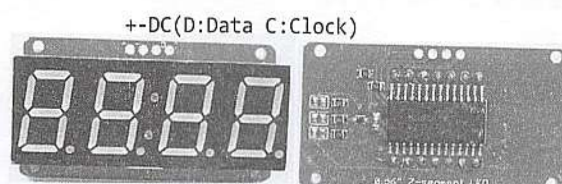
รูปที่ 6.14 จอแสดงผล 7 ส่วน

จอแสดงผล 7 ส่วนมีหลายขนาดและหลากหลายสี เช่น สีแดง สีเขียว หรือสีเหลือง จอแสดงผล 7 ส่วนจะมีการต่อขาร่วม (Common) 2 แบบ คือ

1) Common Anode คือต่อขาร่วม (Common) เข้ากับไฟบวก ส่วนขาอื่นถ้าต้องการให้ติดต้องให้เป็นกราวด์หรือลอจิก 0 จึงจะทำให้หลอดส่วนนั้นสว่าง

2) Common Cathode คือต่อขาร่วมเข้ากับกราวด์ ส่วนขาอื่นถ้าต้องการให้ติดต้องให้เป็นสัญญาณไฟหรือลอจิก 1 จึงจะทำให้หลอดส่วนนั้นสว่าง

เพื่อให้ง่ายในการต่อวงจร รวมทั้งสะดวกต่อการใช้งานและการเขียนโปรแกรม ในที่นี้จะใช้จอแสดงผล 7 ส่วนแบบ I2C Driver HT16K33 สามารถแสดงผลได้ 4 หลัก ขนาด 0.56 นิ้ว ใช้สายสัญญาณควบคุมเพียง 2 เส้น ดังรูปที่ 6.15

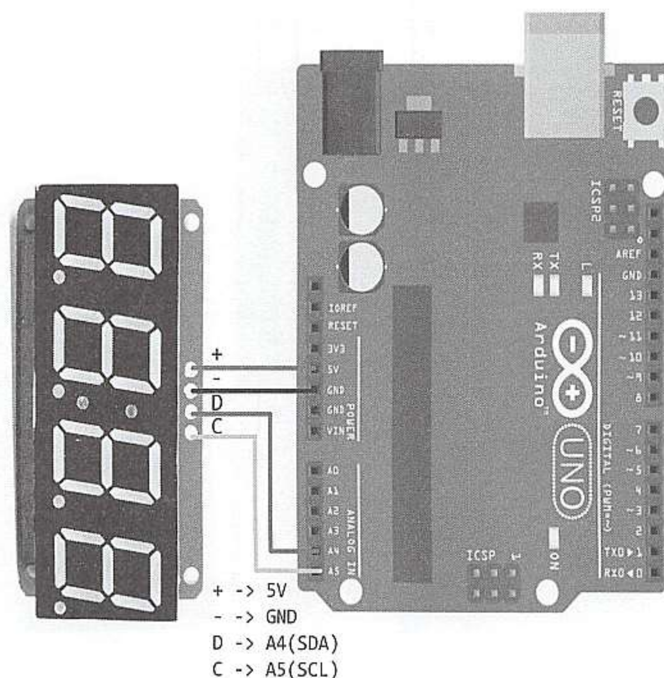


รูปที่ 6.15 จอแสดงผล 7 ส่วนแบบ I2C 4 หลัก

วงจรจอแสดงผล 7 ส่วน 4 หลัก เชื่อมต่อกับบอร์ด Arduino แบบ I2C โดยใช้สายไฟเพียง 2 เส้น คือ ขา D ขาสัญญาณข้อมูลต่อกับขา A4 และขา C ขาสัญญาณนาฬิกาต่อกับขา A5 ในการเขียนโปรแกรมควบคุมการแสดงผลของจอต้องเรียกใช้ไลบรารี #include <Adafruit_LEDBackpack.h> และ #include <Adafruit_GFX.h> สามารถดาวน์โหลดโปรแกรมได้ที่

https://github.com/adafruit/Adafruit-LED-Backpack-Library

https://github.com/adafruit/Adafruit-GFX-Library



รูปที่ 6.16 วงจรการเชื่อมต่อ Arduino กับจอแสดงผล 7 ส่วน 4 หลัก

ฟังก์ชันพื้นฐานที่ใช้ควบคุมการแสดงผลบนจอแสดงผล 7 ส่วนมีดังนี้

matrix.begin(0x70);

หมายถึง กำหนดแอดเดรสของจอแสดงผล 7 ส่วน

matrix.print(1230,DEC);

หมายถึง กำหนดรูปแบบการแสดงตัวเลข 1230 เป็นเลขฐานสิบ

matrix.writeDisplay();

หมายถึง แสดงผลออกจอแสดงผล 7 ส่วน

matrix.writeDigitNum(0,9,true);

หมายถึง แสดงเลข 9 ที่ตำแหน่ง 0 และไม่แสดงจุดออกจอแสดงผล 7 ส่วน

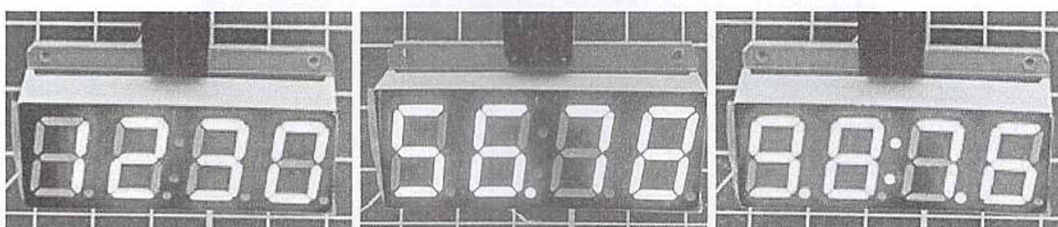
โดยที่	0 หมายถึงตำแหน่ง 0	true แสดงจุด
	9 ตัวเลขที่นำมาแสดง	false ไม่แสดงจุด

ตัวอย่างที่ 6.9 โปรแกรมแสดงตัวเลขบนจอแสดงผล 7 ส่วน

```
#include <Adafruit_GFX.h>                        //เรียกใช้ไลบรารี
#include <Adafruit_LEDBackpack.h>                //เรียกใช้ไลบรารี
Adafruit_7segment matrix=Adafruit_7segment();
void setup()
{
  matrix.begin(0x70);                            //กำหนดแอดเดรส 7 Segment
}
void loop()
{
  matrix.print(1230,DEC);                        //แสดง 1230 เป็นเลขฐานสิบ
  matrix.writeDisplay();                         //แสดงผลออกจอ
  delay(2000);
  matrix.print(56.78);                           //แสดงเลข 56.78
  matrix.writeDisplay();                         //แสดงผลออกจอ
  delay(2000);
  matrix.writeDigitNum(0,9,true);                //แสดงจุดและเลข 9 ที่ตำแหน่ง 0
  matrix.writeDigitNum(1,8,false);               //แสดงเลข 8 ที่ตำแหน่ง 1
  matrix.drawColon(true);                        //แสดงเครื่องหมาย Colon
  matrix.writeDigitNum(3,7,true);                //แสดงจุดและเลข 7 ที่ตำแหน่ง 3
  matrix.writeDigitNum(4,6,false);               //แสดงเลข 6 ที่ตำแหน่ง 4
  matrix.writeDisplay();                         //แสดงผลออกจอ
  delay(2000);
}
```

ผลการรันโปรแกรม

โปรแกรมจะวนรอบแสดงตัวเลข 1230 56.78 และ 9.8:7.6 ออกบนจอแสดงผล 7 ส่วน



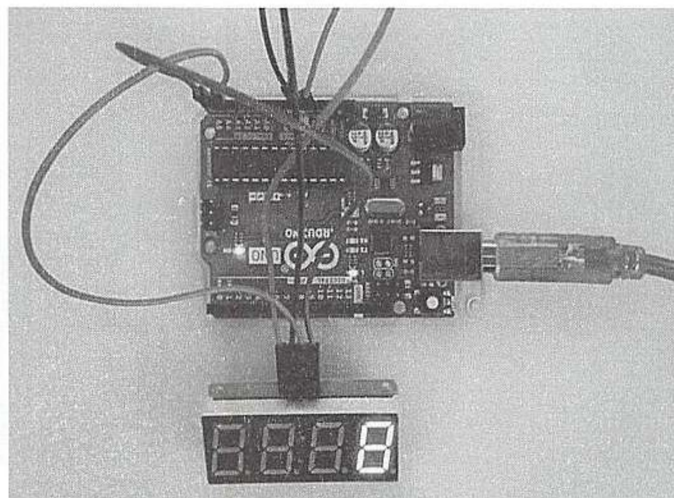
รูปที่ 6.17 การแสดงตัวเลขบนจอแสดงผล 7 ส่วน

ตัวอย่างที่ 6.10 โปรแกรมนับเลข 0–59 ออกบนจอแสดงผล 7 ส่วน

```
#include <Adafruit_GFX.h>                              //เรียกใช้ไลบรารี
#include <Adafruit_LEDBackpack.h>                      //เรียกใช้ไลบรารี
Adafruit_7segment matrix = Adafruit_7segment();
int x;                                                 //ประกาศตัวแปร x แบบ 16 บิต
void setup()
{
matrix.begin(0x70);                                    //กำหนดแอดเดรส 7 Segment
}
void loop()
{
matrix.print(x,DEC);                                   //แสดง x เป็นเลขฐานสิบ
matrix.writeDisplay();                                 //แสดงผลออกจอ
delay(1000);
x++;                                                   //x = x + 1
if (x > 59)                                            //ถ้า x > 59
x = 0;                                                 //ให้ x = 0
}
```

ผลการรันโปรแกรม

โปรแกรมจะวนรอบแสดงตัวเลข 0–59 ออกบนจอแสดงผล 7 ส่วน ที่ตำแหน่ง 3 และตำแหน่ง 4 โดยจะไม่แสดงเลข 0 ในหลักสิบ เมื่อแสดงตัวเลขในหลักหน่วยสำหรับเลข 0–9



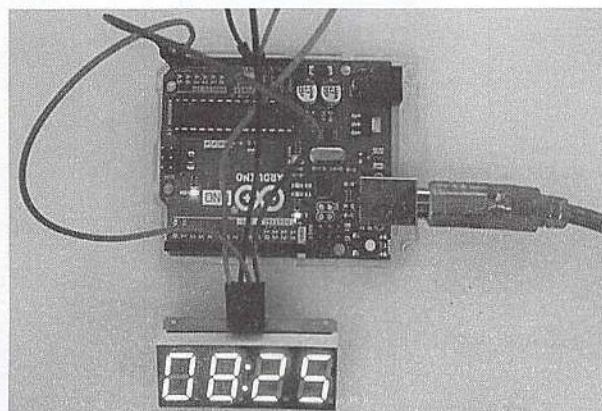
รูปที่ 6.18 การแสดงตัวเลข 0–59 บนจอแสดงผล 7 ส่วน

ตัวอย่างที่ 6.11 โปรแกรมนับและแสดงเลข 00–59 ออกจอแสดงผล 7 ส่วน

```
#include <Adafruit_GFX.h>                              //เรียกใช้ไลบรารี
#include <Adafruit_LEDBackpack.h>                      //เรียกใช้ไลบรารี
Adafruit_7segment matrix = Adafruit_7segment();
int x;
double Tms;
void setup()
{
matrix.begin(0x70);                                    //กำหนดแอดเดรส 7 Segment
}
void loop()
{ if (millis()>Tms)                                    //ถ้า millis() > Tms
  { Tms = 1000+millis();                               //ให้ Tms = 1000 + millis()
    matrix.writeDigitNum(3,x/10,false);                //แสดงหลักสิบของ x ที่ตำแหน่ง 3
    matrix.writeDigitNum(4,x%10,false);                //แสดงหลักหน่วยของ x ที่ตำแหน่ง 4
    matrix.writeDisplay();                             //แสดงผลออก 7 Segment
    x++;
    if (x > 59)                                        //ถ้า x > 59
  x = 0;                                               //ให้ x = 0
  }
}
```

ผลการรันโปรแกรม

โปรแกรมจะวนรอบแสดงตัวเลข 00–59 ออกบนจอแสดงผล 7 ส่วน ที่ตำแหน่ง 3 และตำแหน่ง 4 โดยจะแสดงเลข 0 ในหลักสิบ เมื่อตัวเลขมีค่าไม่ถึง 10



รูปที่ 6.19 การแสดงตัวเลข 00–59 บนจอแสดงผล 7 ส่วน

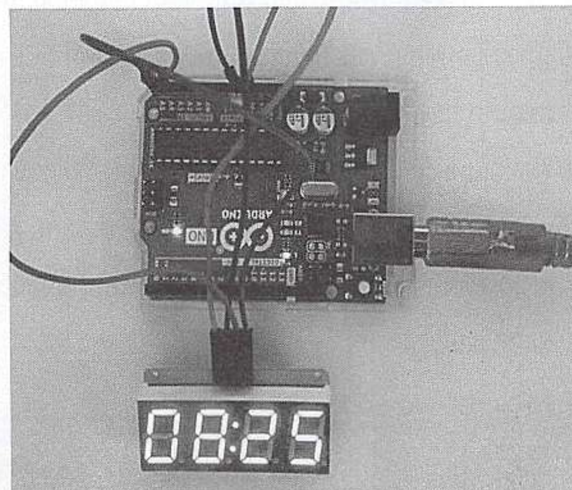
ตัวอย่างที่ 6.12 โปรแกรมแสดงเวลาบนจอแสดงผล 7 ส่วน

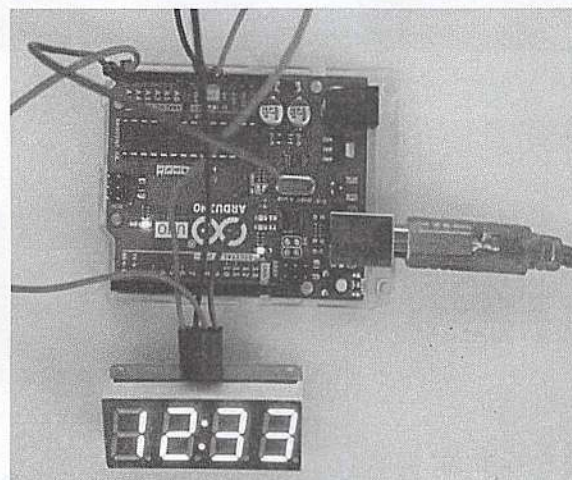
```
#include <Adafruit_GFX.h>                              //เรียกใช้ไลบรารี
#include <Adafruit_LEDBackpack.h>                      //เรียกใช้ไลบรารี
Adafruit_7segment matrix = Adafruit_7segment();
double tm;
int hour, min, sec, colon;
void setup()
{
  matrix.begin(0x70);                                  //กำหนดแอดเดรส 7 Segment
}
void loop()
{
  if(millis() > tm)                                    //ถ้า millis() > tm
    { tm = 1000 + millis();                            //ให้ tm = 1000 + millis()
      colon = !colon;                                  //กลับสถานะ
      sec++;
      if (sec > 59)                                    //ถ้า sec > 59
      { sec = 0;
        min++;
      }
  if (min > 59)                                        //ถ้า min > 59
    { min = 0;
      hour++;
    }
  if (hour > 23)                                       //ถ้า hour > 23
    hour = 0;
  matrix.writeDigitNum(0,hour/10);                     //แสดงหลักสิบของ hour
  matrix.writeDigitNum(1,hour%10);                     //แสดงหลักหน่วยของ hour
  matrix.writeDigitNum(3,min/10,false);                //แสดงหลักสิบของ min
  matrix.writeDigitNum(4,min%10,false);                //แสดงหลักหน่วยของ min
    if(colon == 1)
```

```
      matrix.drawColon(true);                //แสดงเครื่องหมาย Colon
    else
      matrix.drawColon(false);               //ไม่แสดงเครื่องหมาย Colon
    matrix.writeDisplay();                   //แสดงผลออก 7 Segment
  }
}
```

ผลการรันโปรแกรม

โปรแกรมจะวนรอบแสดงเวลา 00:00 ถึง 23:59 บนจอแสดงผล 7 ส่วน และในหลักชั่วโมง นาที จะมีไฟกะพริบทุก ๆ วินาที





รูปที่ 6.20 การแสดงเวลาบนจอแสดงผล 7 ส่วน

6.7 สรุป

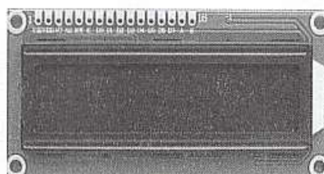
จอแสดงผล LCD สามารถแสดงผลได้ทั้งตัวเลข ตัวอักษร รูปภาพ สัญลักษณ์ กราฟิก ในการเขียนโปรแกรมควบคุมการทำงานของ LCD ต้องเรียกใช้ไลบรารี #include <LiquidCrystal.h> เพื่อที่จะเรียกใช้งานฟังก์ชันต่าง ๆ ในการควบคุม LCD ได้โดยสะดวก

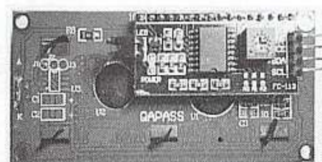
จอแสดงผล LCD แบบ I2C มีข้อดีคือใช้สายสัญญาณในการควบคุม 2 เส้นทำให้สะดวกในการเชื่อมต่อกับไมโครคอนโทรลเลอร์ การเขียนโปรแกรมต้องเรียกใช้ไลบรารี #include <LiquidCrystal_I2C.h> จึงจะสามารถเรียกใช้งานฟังก์ชันต่าง ๆ ได้

I2C (Inter-Integrated Circuit) เป็นการสื่อสารข้อมูลอนุกรมแบบซิงโครนัสเพื่อใช้ติดต่อสื่อสารระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์อินพุตและเอาต์พุต โดยใช้สายสัญญาณเพียง 2 เส้น คือสายสัญญาณข้อมูลแบบอนุกรม (SDA) และสายสัญญาณนาฬิกา (SCL) ซึ่งมีข้อดีคือสามารถเชื่อมต่ออุปกรณ์หลายตัวเข้าด้วยกันโดยใช้สายสัญญาณเพียง 2 เส้น

จอแสดงผล OLED (Organic Light-Emitting Diode) คือไดโอดเปล่งแสงที่ออกแบบเป็นฟิล์มแผ่นบาง ทำปฏิกิริยาคล้ายกับอุปกรณ์เปล่งแสงแบบ LED เม็ดเหลี่ยม มีข้อดีคือสามารถควบคุมตำแหน่งและความสว่างของแสงได้แม่นยำ การเชื่อมต่อจะใช้สายสัญญาณเพียง 2 เส้นคือ SCL ต่อกับขา A5 และ SDA ต่อกับขา A4 ซึ่งเป็นการเชื่อมต่อแบบ I2C

จอแสดงผล 7 ส่วน คือจอแสดงผลตัวเลข 0–9 และตัวอักษรบางตัวที่สามารถแสดงผลได้ การแสดงผลเกิดจากการจัดวางหลอด LED ในแนวยาวและแนวตั้งซึ่งมีทั้งหมด 7 ส่วน คือ a b c d e f g และจุด Dp การใช้งานหากต้องการให้แสดงเลขใดต้องทำให้ LED ในส่วนนั้นติด ในที่นี้ใช้จอแสดงผล 7 ส่วนแบบ I2C Driver HT16K33 ซึ่งสามารถแสดงผลได้ 4 หลัก


LCD 2×16


LCD I2C 2×16




7 Segment I2C

รูปที่ 6.21 จอแสดงผลแบบต่าง ๆ

แบบฝึกหัดบทที่ 6

1. อธิบายขาสัญญาณของจอแสดงผล LCD
2. อธิบายขาสัญญาณของจอแสดงผล LCD แบบ I2C
3. อธิบายการเชื่อมต่อแบบ I2C
4. อธิบายฟังก์ชันหรือคำสั่งต่อไปนี้
   - #include <LiquidCrystal_I2C.h>
   - LiquidCrystal_I2C lcd(0x27,16,2);
   - lcd.begin();
   - lcd.setCursor(0,1);
   - lcd.print("Arduion&Robot");
5. ให้เขียนโปรแกรมแสดงชื่อและเลขที่บนจอแสดงผล LCD
6. ให้เขียนโปรแกรมนาฬิกาที่แสดงหลักนาทีและวินาทีบนจอแสดงผล LCD
7. ให้เขียนโปรแกรมแสดงค่าอุณหภูมิและความชื้นบนจอแสดงผล OLED
8. ให้เขียนโปรแกรมนาฬิกา โดยเมื่อเวลา 8.00 นาฬิกาให้หลอด LED ติดเป็นเวลา 1 นาที

บทที่ 7 ระบบควบคุมตำแหน่งแบบ PID

ระบบควบคุม (Control System) ถูกออกแบบเพื่อควบคุมกระบวนการให้ทำงานตามวัตถุประสงค์ที่ต้องการ ปัญหาหลักของระบบควบคุมคือจะทำอย่างไรให้สัญญาณเอาต์พุตทำงานตามค่าที่ตั้งไว้ มีการตอบสนองที่ทันต่อเวลาและมีค่าผิดพลาดน้อย ระบบควบคุมแบบ PID เป็นระบบควบคุมแบบป้อนกลับชนิดหนึ่งที่ได้รับความนิยมตั้งแต่อดีตจนถึงปัจจุบัน หลักการควบคุมคือจะนำค่าผิดพลาดไปคำนวณโดยใช้สมการทางคณิตศาสตร์ผ่านฟังก์ชัน PID หาค่าเอาต์พุตเพื่อไปควบคุมกระบวนการให้เป็นไปตามค่าที่กำหนด ระบบควบคุมสามารถแบ่งตามสัญญาณป้อนกลับได้ 2 แบบคือ

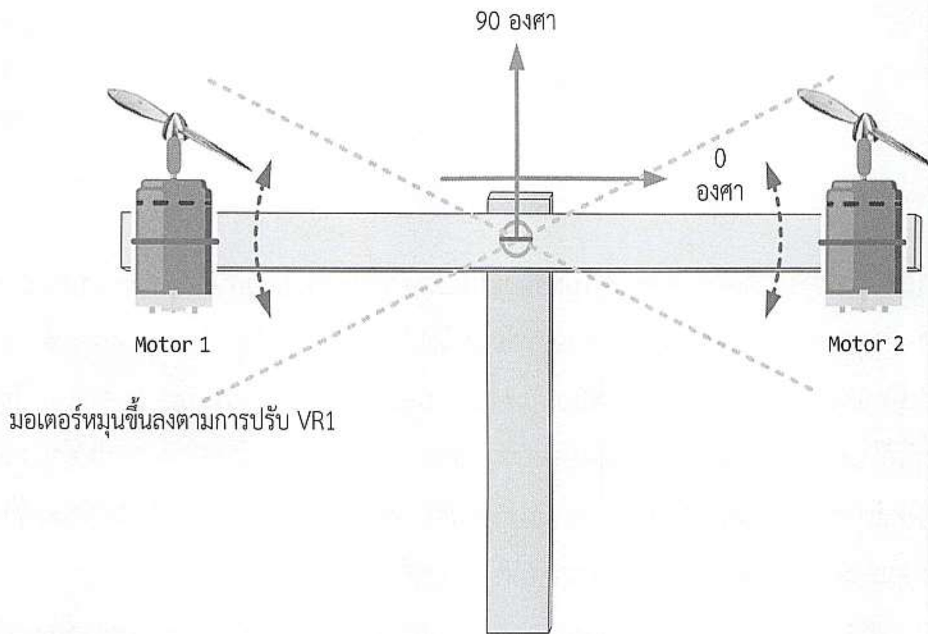
1) ระบบควบคุมแบบเปิด (Open-loop Control) เป็นระบบควบคุมที่ไม่มีการป้อนกลับของสัญญาณเพื่อนำเอาต์พุตมาเปรียบเทียบหรือตรวจสอบความถูกต้องกับสัญญาณอินพุต ข้อดีของระบบควบคุมแบบเปิดคือเป็นระบบที่ควบคุมง่าย ไม่ซับซ้อน ทนทาน และมีราคาถูก

2) ระบบควบคุมแบบปิด (Closed-loop Control) หรือระบบควบคุมแบบป้อนกลับ (Feedback Control) เป็นระบบควบคุมที่มีการป้อนกลับของสัญญาณเอาต์พุตเพื่อนำมาเปรียบเทียบหรือตรวจสอบความถูกต้องกับสัญญาณอินพุต ข้อดีของระบบควบคุมแบบปิดคือเป็นระบบที่ควบคุมให้สัญญาณเอาต์พุตมีความถูกต้องได้มากกว่าระบบควบคุมแบบเปิด แต่ก็มีการทำงานที่ซับซ้อนและมีราคาแพง

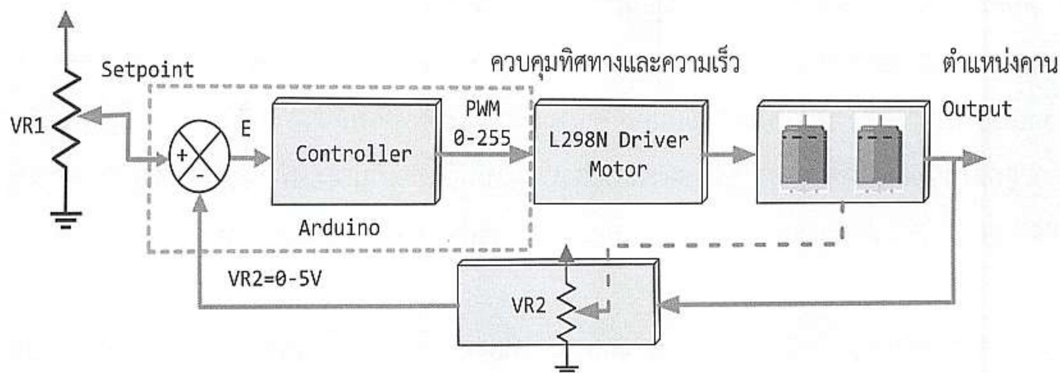
ระบบควบคุมที่ดีควรสามารถทำงานได้ตามค่าที่กำหนดไว้และมีเสถียรภาพ ตัวอย่างในเนื้อหาบทนี้เป็นระบบควบคุมคานให้หมุนไปยังตำแหน่งที่ต้องการ โดยใช้มอเตอร์ 2 ใบพัดเป็นตัวขับเคลื่อนเพื่อเพิ่มความเร็วในการเข้าถึงเป้าหมายหรือตำแหน่งที่กำหนดไว้

7.1 การควบคุมตำแหน่งคาน

ในการควบคุมตำแหน่งคาน ระบบจะพยายามควบคุมคานให้หมุนไปตามตำแหน่งที่ตั้งไว้ โดยจะมีมอเตอร์ 2 ตัวหมุนตามเข็นนาฬิกาหรือทวนเข็มนาฬิกา ทำให้คานหมุนขึ้นลงตามการปรับค่าความต้านทาน VR1 และจะมีความต้านทาน VR2 เป็นอุปกรณ์วัดตำแหน่งของคาน เพื่อส่งสัญญาณให้ไมโครคอนโทรลเลอร์ทำการประมวลผลและส่งสัญญาณควบคุมทิศทางและความเร็วให้คานหมุนไปยังตำแหน่งหรือมุมที่กำหนด

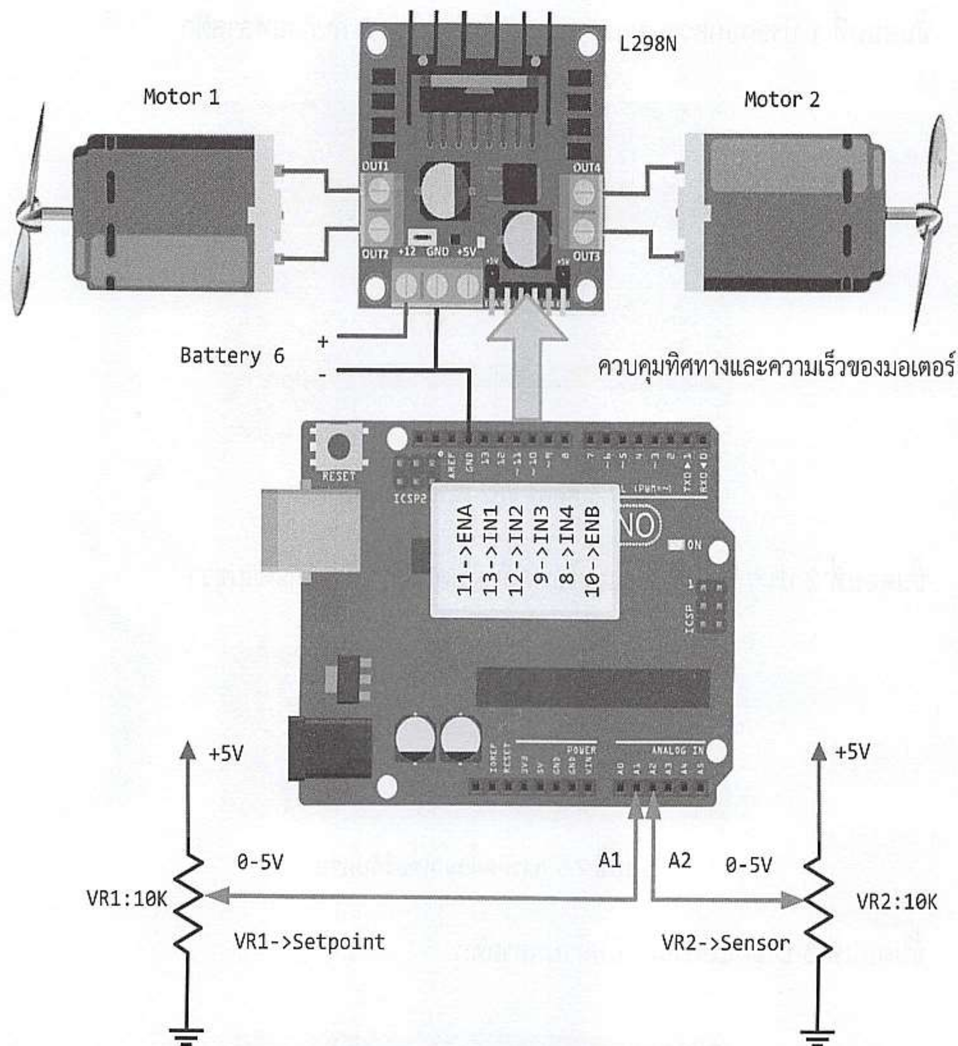


รูปที่ 7.1 การหมุนขึ้นลงของคาน



รูปที่ 7.2 บล็อกไดอะแกรมการควบคุมตำแหน่งของคาน

วงจรควบคุมตำแหน่งของคานจะมีความต้านทาน VR1 เป็นตัวปรับค่าตำแหน่งที่ต้องการ (Setpoint) และความต้านทาน VR2 เป็นอุปกรณ์วัดตำแหน่งคาน โดยจะส่งสัญญาณให้ไมโครคอนโทรลเลอร์ที่ขา A1 และ A2 เพื่อทำการประมวลผลและส่งสัญญาณไปควบคุมมอเตอร์ให้หมุนไปตามตำแหน่งที่กำหนดไว้



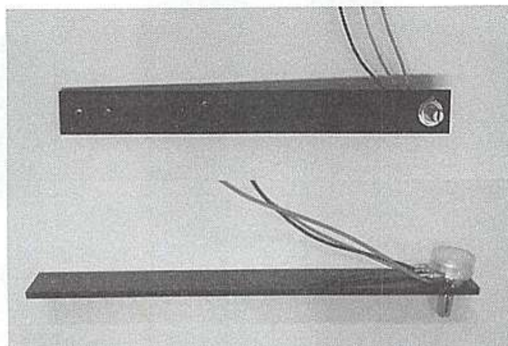
รูปที่ 7.3 วงจรควบคุมตำแหน่งของคาน

จากรูปที่ 7.3 VR1 เป็นตัวปรับค่าตำแหน่งที่ต้องการ (Setpoint) VR2 เป็นอุปกรณ์วัดตำแหน่งของคาน ส่งสัญญาณให้ไมโครคอนโทรลเลอร์ที่ขา A1 และ A2 ของบอร์ด Arduino เพื่อทำการประมวลผล แล้วส่งสัญญาณควบคุมแรงดัน PWM ออกทางขา 10 และขา 11 ส่วนขา 12 13 ควบคุมทิศทางมอเตอร์ 1 ขา 8 ขา 9 ควบคุมทิศทางมอเตอร์ 2 โดยที่บอร์ด L298N เป็นวงจรขยายสัญญาณเพื่อควบคุมความเร็วและทิศทางของมอเตอร์ตัวที่ 1 และ 2 เพื่อให้คานหมุนไปยังตำแหน่งหรือมุมที่กำหนด

7.2 การสร้างชุดควบคุมตำแหน่งของคาน

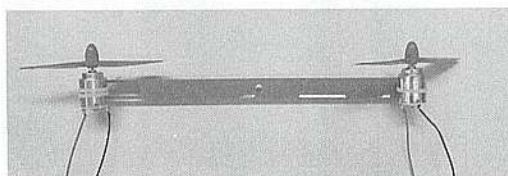
ในการทดลองนี้จะทำการสร้างชุดควบคุมตำแหน่งของคานแบบ 2 ใบพัด ใช้มอเตอร์ 2 ตัว มีความต้านทานแบบปรับค่าได้ VR1 เป็นตัวกำหนดตำแหน่งของคาน ความต้านทานแบบปรับค่าได้ VR2 เป็นตัวตรวจวัดตำแหน่งของคาน โดยมอเตอร์ทั้งสองตัวเป็นตัวควบคุมตำแหน่งของคาน การสร้างชุดทดลองมีขั้นตอนดังนี้

ขั้นตอนที่ 1 ประกอบความต้านทานแบบปรับค่าได้ VR2 กับก้านพลาสติก



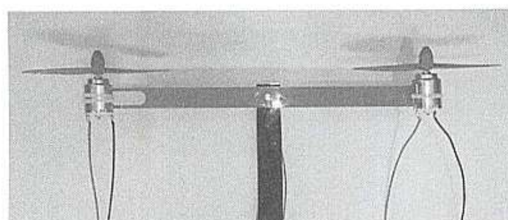
รูปที่ 7.4 ติดตั้ง VR2 เข้ากับก้านพลาสติก

ขั้นตอนที่ 2 ประกอบมอเตอร์และใบพัดเข้ากับคานด้านซ้ายและด้านขวา



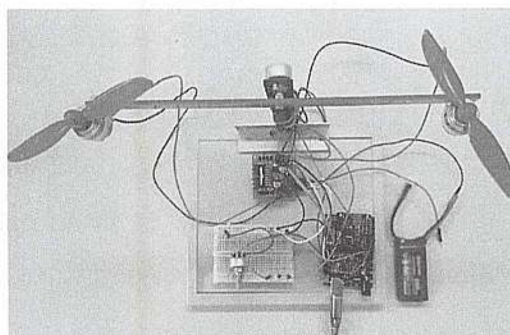
รูปที่ 7.5 การติดตั้งมอเตอร์กับคาน

ขั้นตอนที่ 3 ประกอบคานเข้ากับก้านพลาสติก



รูปที่ 7.6 ติดตั้งคานเข้ากับก้านพลาสติก

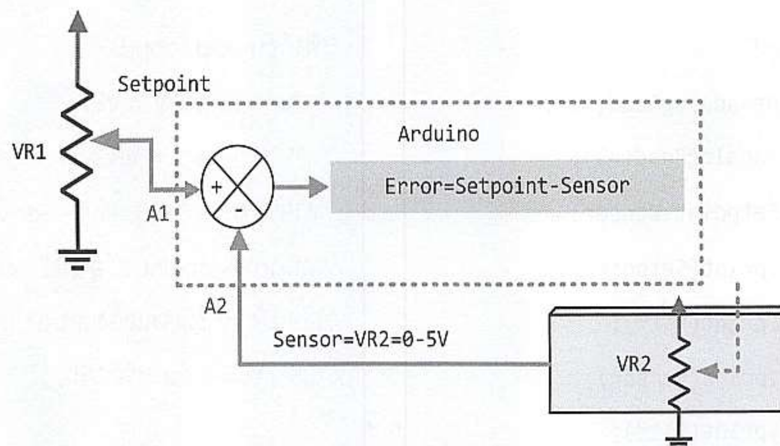
ขั้นตอนที่ 4 ติดตั้งชุดคานเข้ากับฐานไม้หรือโต๊ะ แล้วต่อสายสัญญาณ VR1 VR2 วงจรขยายสัญญาณและมอเตอร์ ตามวงจรรูปที่ 7.3



รูปที่ 7.7 ชุดควบคุมตำแหน่งของคาน

7.3 ค่าผิดพลาด

ค่าผิดพลาด (Error) คือค่าความแตกต่างระหว่างค่าที่ตั้งไว้กับค่าที่วัดได้ ซึ่งปัญหาของระบบควบคุมคือจะทำอย่างไรกับค่าผิดพลาดที่เกิดขึ้น แล้วจะส่งสัญญาณไปควบคุมกระบวนการอย่างไรเพื่อให้ค่าผิดพลาดลดลง การหาค่าผิดพลาดแสดงดังรูปที่ 7.8



รูปที่ 7.8 การหาค่าผิดพลาด

โดยสามารถคำนวณหาค่าผิดพลาดได้ดังนี้

```
Error=Setpoint-Sensor;
```

Setpoint คือ ค่าตำแหน่งที่ตั้งไว้จากการปรับค่าความต้านทาน VR1 ซึ่งมีค่า 0–5 โวลต์ หรือ 0–1023 เมื่อแปลงเป็นสัญญาณดิจิทัล

Sensor คือ ค่าตำแหน่งที่วัดได้ โดยวัดค่าจากความต้านทาน VR2 ซึ่งจะหมุนตามคาน มีค่า 0–5 โวลต์ หรือ 0–1023 เมื่อแปลงเป็นสัญญาณดิจิทัล

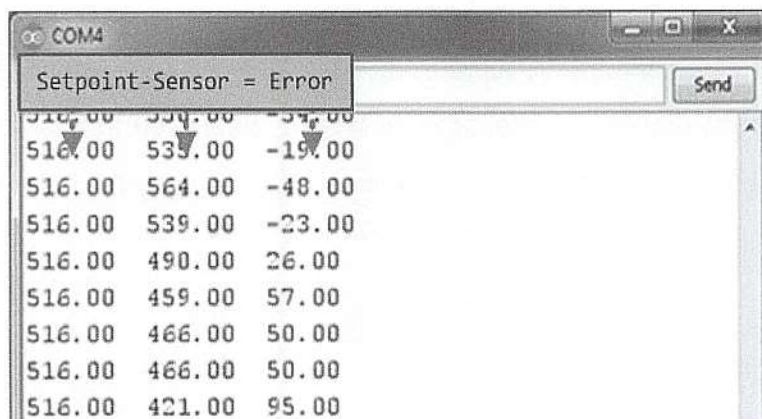
Error คือ ค่าความผิดพลาด ซึ่งจะมีค่าอยู่ระหว่าง –1023 ถึง +1023 เนื่องจากค่า Setpoint และ Sensor มีค่าอยู่ในช่วง 0–1023 ในการทดลอง Error จะมีค่าอยู่ในช่วง –200 ถึง +200 การประมาณค่าผิดพลาดได้จะทำให้สามารถกำหนดค่าเกณฑ์ Ki Kp และ Kd ของการควบคุมแบบ PID ได้ง่ายขึ้น

ตัวอย่างที่ 7.1 โปรแกรมหาค่าความผิดพลาดของระบบควบคุม

```
int Setpoint,Sensor,Error;              //ประกาศตัวแปร
void setup()                            //ฟังก์ชัน void setup()
{ Serial.begin(9600);                   //กำหนดการรับส่งข้อมูลแบบอนุกรม
}
void loop()                             //ฟังก์ชัน void loop()
{ Setpoint=analogRead(A1);              //อ่านค่า Setpoint = VR1
  Sensor=analogRead(A2);                //อ่านค่า Sensor = VR2
  Error=Setpoint-Sensor;                //หาค่า Error = Setpoint – Sensor
  Serial.print(Setpoint);               //แสดงค่า Setpoint หรือ VR1
  Serial.print("\t");                   //เลื่อนตำแหน่งการแสดงผลบนจอภาพ
  Serial.print(Sensor);                 //แสดงค่า Sensor หรือ VR2
  Serial.print("\t");
  Serial.println(Error);                //แสดงค่า Error
  delay(1000);
}
```

ผลการรันโปรแกรม

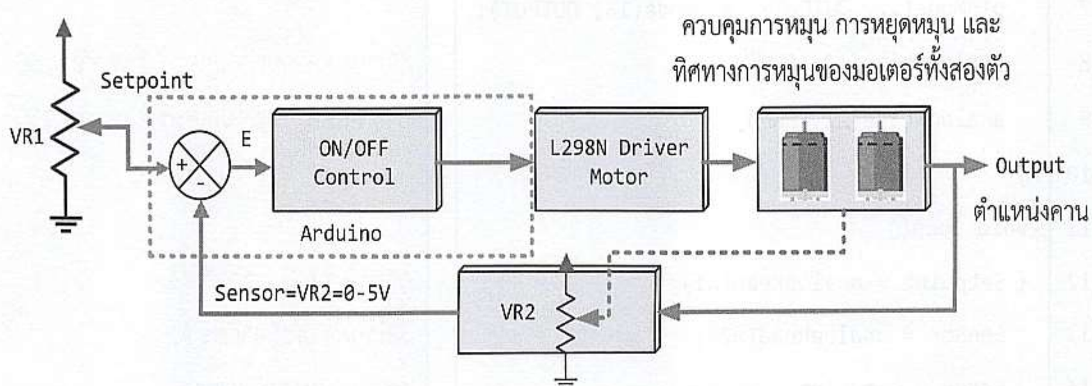
โปรแกรมจะแสดงค่า Setpoint Sensor และ Error ออกบนจอภาพตามการปรับค่าความต้านทาน VR1 และการหมุนหรือการแกว่งของคาน ซึ่ง Error จะมีค่าเป็นบวกเมื่อ Setpoint มากกว่า Sensor และจะเป็นลบเมื่อ Setpoint น้อยกว่า Sensor โดยค่า Error จากการทดลองจะอยู่ในช่วง –200 ถึง +200 เมื่อปรับตำแหน่งของ VR1 และ VR2 ไปในทิศทางเดียวกันค่าผิดพลาดต้องลดลง หากไม่เป็นไปตามนี้ต้องกลับสายสัญญาณ VR ตัวใดตัวหนึ่ง ผลการรันโปรแกรมแสดงดังรูปที่ 7.9



รูปที่ 7.9 การแสดงค่าผิดพลาดของระบบควบคุม

7.4 การควบคุมตำแหน่งของคานแบบเปิด-ปิด

ระบบควบคุมแบบเปิด-ปิด (ON/OFF Control System) ระบบจะควบคุมให้เอาต์พุตทำงานเพียง 2 สภาวะคือเปิดและปิด หรือ ON และ OFF การควบคุมตำแหน่งของคานแบบเปิด-ปิดจะคำนวณหาค่าผิดพลาดจากความแตกต่างของค่าความต้านทาน VR1 และ VR2 ถ้าค่าผิดพลาดน้อยกว่าค่าที่กำหนดไว้จะควบคุมให้มอเตอร์หยุดทำงาน แต่ถ้ามีค่าผิดพลาดมากกว่าค่าที่กำหนดไว้ จะควบคุมให้มอเตอร์ทั้งสองตัวทำงานเพื่อให้คานเอียงไปด้านซ้ายหรือด้านขวา (ในทิศตามเข็มหรือทวนเข็มนาฬิกา) เป็นไปตามค่าผิดพลาดที่คำนวณได้ว่าเป็นลบหรือบวก



รูปที่ 7.10 การควบคุมตำแหน่งของคานแบบเปิด-ปิด

Setpoint คือ ค่าตำแหน่งที่ตั้งไว้จากการปรับค่าความต้านทาน VR1 ซึ่งมีค่า 0–5 โวลต์ หรือ 0–1023 เมื่อแปลงเป็นสัญญาณดิจิทัล

Sensor คือ ค่าตำแหน่งที่วัดได้ โดยวัดค่าจากความต้านทาน VR2 ซึ่งจะหมุนตามคาน มีค่า 0–5 โวลต์ หรือ 0–1023 เมื่อแปลงเป็นสัญญาณดิจิทัล

Error คือ ค่าผิดพลาด ซึ่งค่าที่เป็นบวกหรือลบจะมีผลต่อทิศทางการหมุนของมอเตอร์

ในระบบควบคุมแบบเปิด-ปิด หรือ ON/OFF Control System ไมโครคอนโทรลเลอร์จะควบคุมให้มอเตอร์ทำงานหรือหยุดทำงานตามเงื่อนไขการควบคุม ซึ่งจะไม่สามารถปรับความเร็วของมอเตอร์ได้ โดยมอเตอร์ทั้งสองตัวจะหมุนในทิศทางตรงข้ามกัน และสัญญาณที่ส่งไปควบคุมความเร็วของมอเตอร์มีค่าอยู่ในช่วง 0–255 ซึ่งเป็นการควบคุมแบบ PWM

L298N ทำหน้าที่ขยายสัญญาณไฟฟ้า ควบคุมความเร็วและทิศทางของมอเตอร์ทั้งสองตัวตามสัญญาณที่ส่งมาจากไมโครคอนโทรลเลอร์

Output คือ ค่าตำแหน่งหรือมุมของคาน ซึ่งจะมีการแกว่งเป็นปกติของการควบคุมแบบเปิด-ปิด

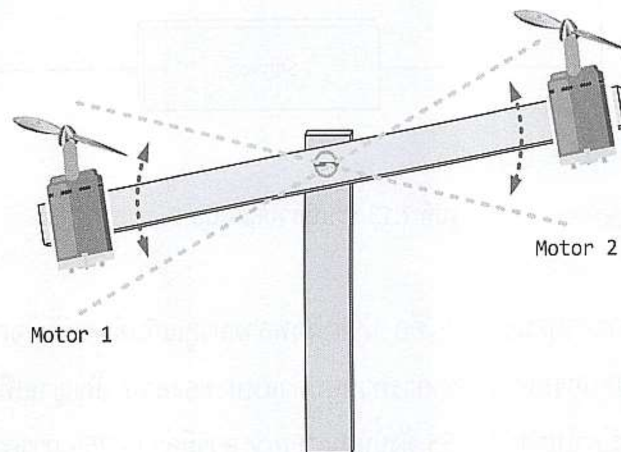
ตัวอย่างที่ 7.2 โปรแกรมควบคุมตำแหน่งคานแบบเปิด-ปิด

```
int Setpoint, Sensor;                          //ประกาศตัวแปร
int Error, PWM=150;                            //กำหนดความเร็วของมอเตอร์
void setup()
{ Serial.begin(9600);                          //กำหนดการรับส่งข้อมูลแบบอนุกรม
  pinMode(8, OUTPUT); pinMode(9, OUTPUT);
  pinMode(10, OUTPUT); pinMode(11, OUTPUT);
  pinMode(12, OUTPUT); pinMode(13, OUTPUT);
  analogWrite(10, PWM);                        //กำหนดความเร็วมอเตอร์ 1 เท่ากับ 150
  analogWrite(11, PWM);                        //กำหนดความเร็วมอเตอร์ 2 เท่ากับ 150
}
void loop()
{ Setpoint = analogRead(A1);                   //อ่านค่า VR1 จากขา A1
  Sensor = analogRead(A2);                     //อ่านค่า VR2 จากขา A2
  Error = Setpoint - Sensor;                   //คำนวณหาค่าผิดพลาด
  if (Error >-10 && Error < 10)                //ถ้า Error อยู่ในช่วง –9 ถึง +9
    { digitalWrite(8, LOW);                    //ให้มอเตอร์ทั้งสองตัวหยุดทำงาน
      digitalWrite(9, LOW);
      digitalWrite(12, LOW);
      digitalWrite(13, LOW);
    }
if  (Error <= -10)                             //ถ้า Error น้อยกว่าหรือเท่ากับ –10
  { digitalWrite(8, HIGH);                     //ให้มอเตอร์ 2 หมุนตามเข็มนาฬิกา
    digitalWrite(9, LOW);
    digitalWrite(12, HIGH);                    //มอเตอร์ 1 หมุนทวนเข็มทำให้คานเอียงซ้าย
    digitalWrite(13, LOW);
  }
if (Error >= 10)                               //ถ้า Error มากกว่าหรือเท่ากับ 10
  { digitalWrite(8, LOW);                      //ให้มอเตอร์ 2 หมุนทวนเข็มนาฬิกา
    digitalWrite(9, HIGH);
    digitalWrite(12, LOW);
    digitalWrite(13, HIGH);                    //มอเตอร์ 1 หมุนตามเข็มทำให้คานเอียงขวา
  }
Serial.print(Setpoint);                        //แสดงค่า Setpoint
```

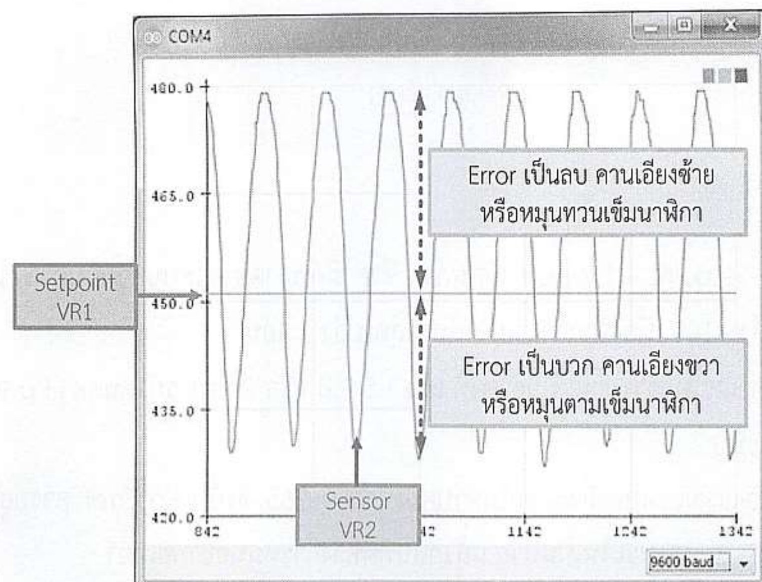
```
Serial.print("\t");
Serial.println(Sensor);          //แสดงค่า Sensor
}
```

ผลการรันโปรแกรม

ถ้าค่าผิดพลาดหรือ Error มากกว่าหรือเท่ากับ 10 จะทำให้คานเอียงไปทางขวา ถ้า Error น้อยกว่าหรือเท่ากับ –10 จะทำให้คานเอียงไปทางซ้าย แต่ถ้า Error มากกว่า –10 และน้อยกว่า 10 (–9 ถึง +9) มอเตอร์จะหยุดหมุน คานจะหมุนเอียงซ้ายขวาตามการปรับค่า Setpoint หรือ VR1 การควบคุมตำแหน่งของคานก็คือการแกว่งขึ้นลงของคาน ซึ่งเป็นค่าปกติของการควบคุมแบบเปิด-ปิด แสดงดังรูปที่ 7.11 และ 7.12



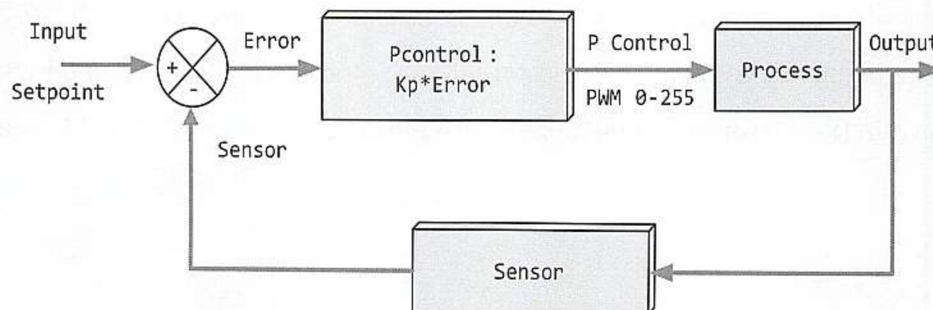
รูปที่ 7.11 คานหมุนขึ้นลงตามการปรับค่า VR1



รูปที่ 7.12 กราฟแสดงการควบคุมตำแหน่งของคานแบบเปิด-ปิด

7.5 การควบคุมแบบ P

การควบคุมแบบ P (Proportional) เป็นการควบคุมแบบอัตราส่วนตามค่าเกณฑ์ Kp ที่กำหนดไว้ โดยเอาต์พุตของการควบคุมแบบ P จะขึ้นอยู่กับค่าผิดพลาดคูณกับเกณฑ์การขยาย Kp ในการปรับค่าเกณฑ์การขยาย ถ้า Kp มีค่ามาก ทำให้เกิดสัญญาณเอาต์พุตเกินค่าเป้าหมาย แต่ถ้า Kp มีค่าน้อย ทำให้เอาต์พุตมีค่าน้อยกว่าเป้าหมายที่กำหนดไว้



รูปที่ 7.13 ระบบควบคุมแบบ P

หลักการของการควบคุมแบบ P คือ นำค่าผิดพลาดมาคูณกับค่าคงที่หรือค่าเกณฑ์การขยาย Kp แล้วส่งสัญญาณไปควบคุมกระบวนการ การควบคุมตำแหน่งของคาน สัญญาณที่ส่งไปควบคุมมอเตอร์เป็นสัญญาณ PWM จะมีค่าในช่วง 0–255 ดังนั้น Kp*Error ควรมีค่า 0–255 การควบคุมแบบ P สามารถคำนวณหาค่าได้ดังนี้

```
Error=Setpoint-PV;
P=Kp*Error;
```

Error = Setpoint – PV คือ ค่าผิดพลาด ซึ่งค่าผิดพลาดของระบบควบคุมตำแหน่งของคานมีค่าประมาณ –100 ถึง +100 เนื่องจากคานหมุนได้มากสุดไม่ถึง 1 รอบ

Kp คือ ค่าเกณฑ์การขยาย เป็นค่าคงที่ เช่น 1.5 1.8 หรือ 2 ซึ่งการกำหนดค่า Kp จะพิจารณาจากค่าผิดพลาดของระบบ

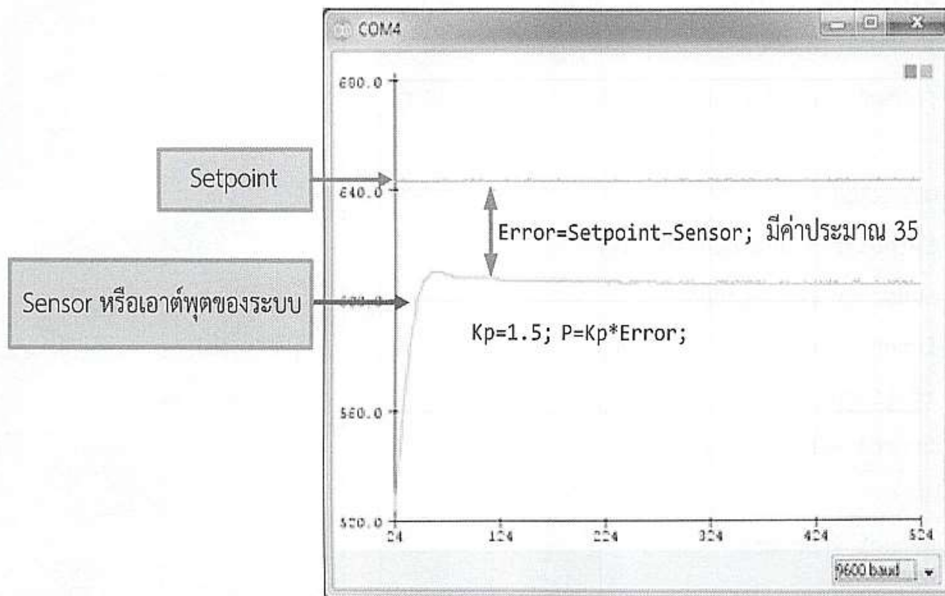
P คือ สัญญาณเอาต์พุตที่ส่งออกไปควบคุม มีค่า 0–255 ดังนั้น Kp*Error ควรอยู่ในช่วง 0–255 หาก Error มีค่าเป็นลบต้องปรับให้เป็นบวก แล้วกลับทิศทางการหมุนของมอเตอร์

ตัวอย่างที่ 7.3 โปรแกรมควบคุมตำแหน่งคานแบบ P

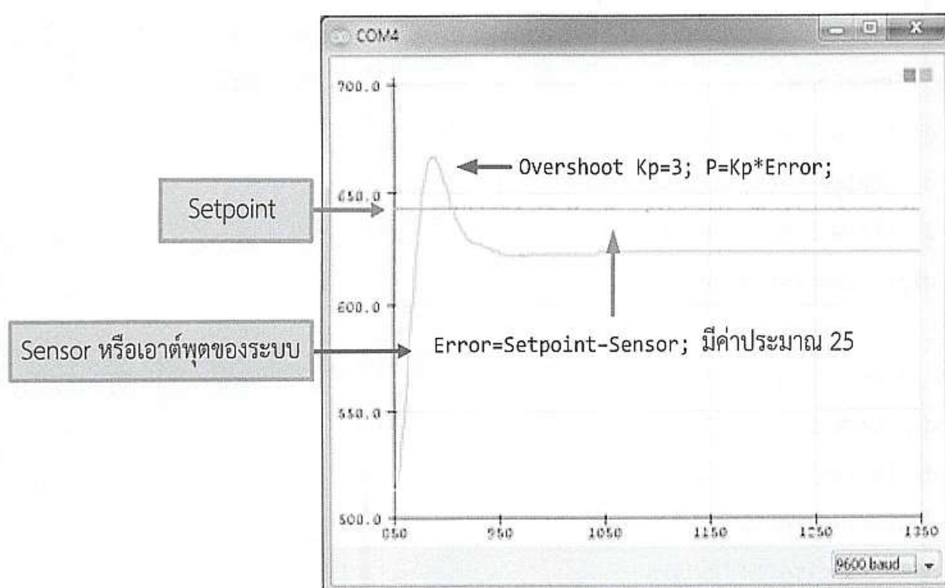
```
float Setpoint,Sensor;
float Error,Error1,P,Kp=1.5;
void setup()
{ Serial.begin(9600);
  pinMode(8, OUTPUT);   pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);  pinMode(13, OUTPUT);
}
void loop()
{ Setpoint = analogRead(A1);          //อ่านค่า VR1 = Setpoint
  Sensor = analogRead(A2);            //อ่านค่า VR2 = Sensor
  Error = Setpoint - Sensor;          //หาค่า Error
  if (Error < 0)                      //ถ้า Error เป็นลบ
  Error1 = Error * (-1);              //ให้ Error1 มีค่าเป็นบวก
else
  Error1 = Error;
P = Kp * Error1;                      //คำนวณหาค่า P Control
if (P > 240) P = 240;
if (P < 0)   P = 0;
analogWrite(10,P);                    //ควบคุมความเร็วมอเตอร์ 1
analogWrite(11,P);                    //ควบคุมความเร็วมอเตอร์ 2
if (Error > 0)                        //ถ้า Error > 0 ให้หมุนคานลง
{ digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  digitalWrite(12, LOW);
  digitalWrite(13, HIGH);
}
if (Error < 0)                        //ถ้า Error < 0 ให้หมุนคานขึ้น
{ digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  digitalWrite(12, HIGH);
  digitalWrite(13, LOW);
} Serial.print(Setpoint);             //แสดงค่า Setpoint ที่ตั้งไว้
  Serial.print("\t");
  Serial.println(Sensor);             //แสดงค่า Sensor ที่วัดได้
}
```

ผลการรันโปรแกรม

โปรแกรมหาค่าผิดพลาดจาก Error = Setpoint – Sensor จากนั้นคำนวณหาค่าเอาต์พุตด้วยการควบคุมแบบ P คือ P = Kp*Error แล้วส่งค่าสัญญาณเอาต์พุตแบบ PWM ไปควบคุมมอเตอร์ทั้งสองตัวให้หมุนไปยังตำแหน่งเป้าหมายที่ตั้งไว้ ซึ่งอาจมีค่าผิดพลาดเล็กน้อยตามการปรับค่า Kp การควบคุมแบบ P คานจะไม่แกว่งหรือมีความเสถียรมากกว่าการควบคุมแบบเปิด-ปิด แต่จะมีค่าผิดพลาดในระยะคงตัว



รูปที่ 7.14 กราฟผลตอบสนองของระบบควบคุมตำแหน่งแบบ P โดยที่ Kp = 1.5

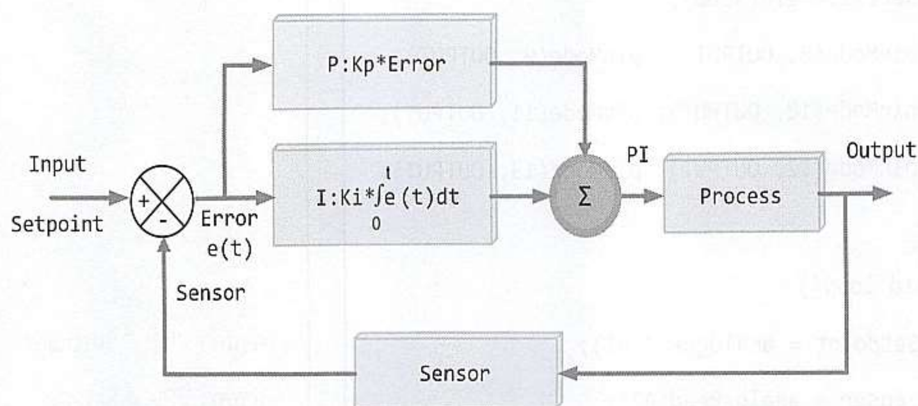


รูปที่ 7.15 กราฟผลตอบสนองของระบบควบคุมตำแหน่งแบบ P โดยที่ Kp = 3

7.6 การควบคุมแบบ PI

การควบคุมแบบ I (Integral) เป็นการอินทิเกรตสัญญาณผิดพลาดคูณกับค่าเกณฑ์ Ki หรือรวมเอาค่าผิดพลาดทั้งหมดมาคูณค่า Ki เพื่อเร่งเวลาการเข้าสู่ค่าเป้าหมายและลดค่าผิดพลาดที่เหลืออยู่จากการใช้ฟังก์ชัน P ซึ่งต้องใช้ร่วมกับการควบคุมแบบ P

การควบคุมแบบ PI คือ การรวมสัญญาณของการควบคุมแบบ P และ I เข้าไว้ด้วยกันแล้วส่งสัญญาณเอาต์พุตไปควบคุมการหมุนมอเตอร์หรือตำแหน่งของคาน



รูปที่ 7.16 ระบบควบคุมแบบ PI

การควบคุมแบบ PI สามารถคำนวณหาค่าต่าง ๆ ได้ดังนี้

```
Error=Setpoint-Sensor;
Ierror=Ierror+Error;
I=Ki*Ierror;
PI=Ki*Ierror+Kp*Error;
```

Ki = คือ ค่าเกณฑ์ เป็นค่าคงที่

Ierror = Ierror + Error คือ ผลรวมของค่าผิดพลาดทั้งหมดจากเวลาเริ่มต้นถึงเวลาขณะที่รันโปรแกรม (0–t)

I คือ เอาต์พุตของการควบคุมแบบ I

PI คือ เอาต์พุตของการควบคุมแบบ P และ I รวมกัน มีค่าอยู่ในช่วง 0–255

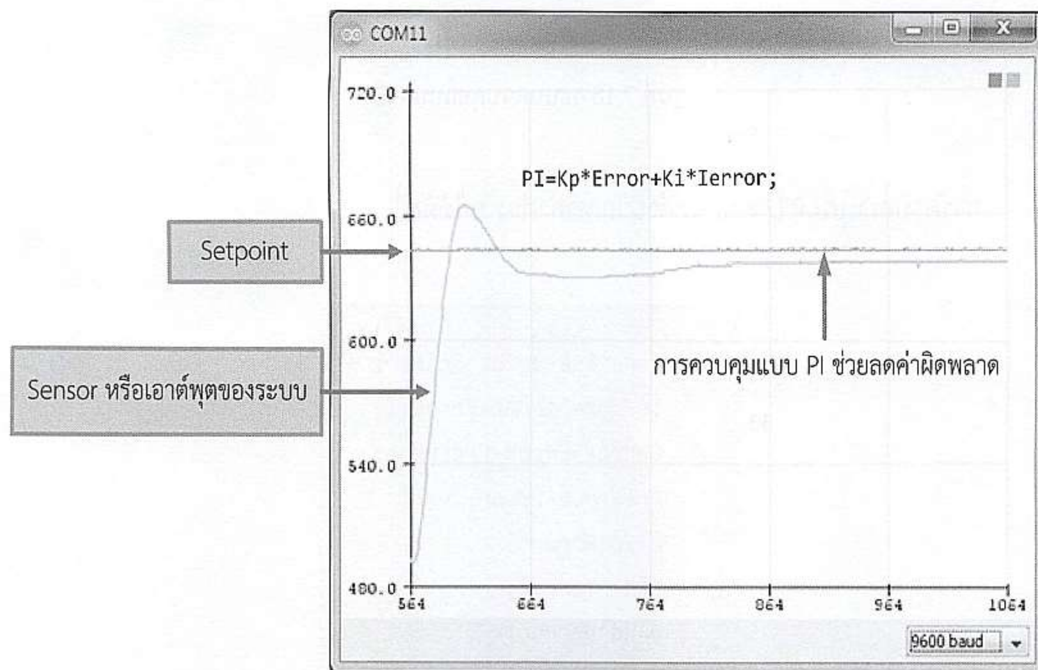
ตัวอย่างที่ 7.4 การควบคุมตำแหน่งคานแบบ PI

```
float Setpoint, Sensor;
float Error, Error1, Ierror;
float Kp = 1.5, Ki = 0.005;
float P, I, PIcontrol;
void setup()
{ Serial.begin(9600);
  pinMode(8, OUTPUT);  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT); pinMode(11, OUTPUT);
  pinMode(12, OUTPUT); pinMode(13, OUTPUT);
}
void loop()
{ Setpoint = analogRead(A1);                 //อ่านค่า VR1 = Setpoint
  Sensor = analogRead(A2);                   //อ่านค่า VR2 = Sensor
  Error = Setpoint - Sensor;                 //หาค่า Error
if(Error<0) Error1=Error*(-1);               //ให้ Error1 มีค่าเป็นบวก
else Error1 = Error;
Ierror = Ierror + Error;                     //หาค่า Error สะสม
P = Kp * Error1;                             //คำนวณหาค่า P Control
I = Ki * Ierror;                             //คำนวณหาค่า I Control
if (I > 200) I = 200;
PIcontrol = P+I;                             //คำนวณหาค่า PI Control
if (PIcontrol>240) PIcontrol=240;
if (PIcontrol < 0) PIcontrol=0;
analogWrite(10, PIcontrol);                  //ควบคุมความเร็วมอเตอร์ 1
analogWrite(11, PIcontrol);                  //ควบคุมความเร็วมอเตอร์ 2
if (Error > 0)                               //ถ้า Error > 0 ให้หมุนคานลง
{ digitalWrite(8, LOW); digitalWrite(9, HIGH);
  digitalWrite(12, LOW); digitalWrite(13, HIGH);
}
if (Error < 0)                               //ถ้า Error < 0 ให้หมุนคานขึ้น
{ digitalWrite(8, HIGH); digitalWrite(9, LOW);
```

```
  digitalWrite(12, HIGH); digitalWrite(13, LOW);
}
  Serial.print(Setpoint);          //แสดงค่า Setpoint ที่ตั้งไว้
  Serial.print("\t");
  Serial.println(Sensor);          //แสดงค่า Sensor ที่วัดได้
}
```

ผลการรันโปรแกรม

โปรแกรมจะหาค่าผิดพลาดของตำแหน่ง จากนั้นคำนวณหาค่าเอาต์พุตของการควบคุมแบบ P และ I แล้วนำสัญญาณเอาต์พุตมารวมกัน แล้วส่งสัญญาณเพื่อไปควบคุมมอเตอร์ให้หมุนไปยังตำแหน่งที่ตั้งไว้ ซึ่งอาจมีค่าผิดพลาดและแกว่งเล็กน้อยจึงต้องปรับค่าเกณฑ์ Kp และ Ki ให้เหมาะสม การควบคุมแบบ PI จะช่วยลดเวลาในการเข้าสู่ค่าตำแหน่งเป้าหมายและลดค่าผิดพลาดในระยะคงตัว

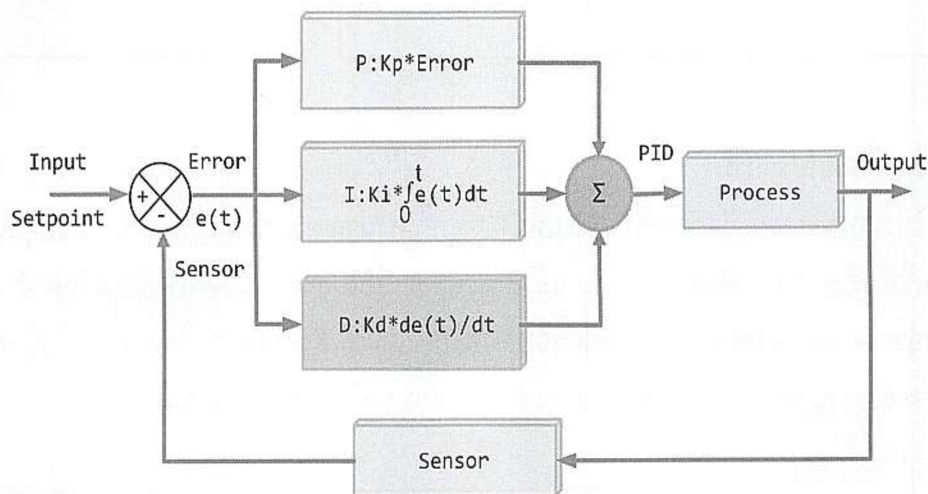


รูปที่ 7.17 กราฟผลตอบสนองของระบบควบคุมตำแหน่งแบบ PI

7.7 การควบคุมแบบ PID

การควบคุมแบบ D (Derivative) เป็นการหาค่าความแตกต่างของสัญญาณผิดพลาดปัจจุบันกับค่าความผิดพลาดสะสม แล้วนำมาคูณกับค่าเกณฑ์ Kd เพื่อลดค่าผิดพลาดในระยะคงตัว

การควบคุมแบบ PID คือ การรวมสัญญาณของการควบคุมแบบ P I และ D เข้าไว้ด้วยกัน แล้วส่งสัญญาณเอาต์พุตไปควบคุมการหมุนมอเตอร์หรือตำแหน่งของคานตามค่าเป้าหมายที่กำหนดไว้



รูปที่ 7.18 ระบบควบคุมแบบ PID

การควบคุมแบบ PID สามารถคำนวณหาค่าต่าง ๆ ได้ดังนี้

```
Error=Setpoint-Sensor;
Ierror=Ierror+Error;
Derror=Error-PreError;
PreError=Error;
P=Kp*Error;
I=Ki*Ierror;
D=Kd*Derror;
PID=P+I+D;
```

Error = Setpoint – Sensor คือ ค่าผิดพลาด ซึ่งค่าผิดพลาดของระบบควบคุมตำแหน่งของคานมีค่าประมาณ –100 ถึง +100 เนื่องจากคานหมุนได้มากสุดไม่ถึง 1 รอบ

Ierror = Ierror + Error คือ ผลรวมของค่าผิดพลาดทั้งหมดจากเวลาเริ่มต้นถึงเวลาขณะที่รันโปรแกรม (0–t)

Derror = Error – PreError หรือผลต่างของค่าผิดพลาดปัจจุบันกับค่าผิดพลาดก่อนหน้านั้น

Kp คือ ค่าเกณฑ์ เป็นค่าคงที่ เช่น 1.5 1.8 หรือ 2 ซึ่งการกำหนดค่า Kp จะพิจารณาจากค่าผิดพลาดของระบบ

Ki = คือ ค่าเกณฑ์ เป็นค่าคงที่ของการควบคุมแบบ I

Kd = คือ ค่าเกณฑ์ เป็นค่าคงที่ของการควบคุมแบบ D

P คือ สัญญาณเอาต์พุตของการควบคุมแบบ P โดยที่ P = Kp*Error

I คือ สัญญาณเอาต์พุตของการควบคุมแบบ I โดยที่ I = Ki*Ierror

D คือ สัญญาณเอาต์พุตของการควบคุมแบบ D โดยที่ D = Kd*Derror

PID คือ สัญญาณเอาต์พุตของการควบคุมแบบ PID โดยที่ PID = Kp*Error + Ki*Ierror + Kd*Derror

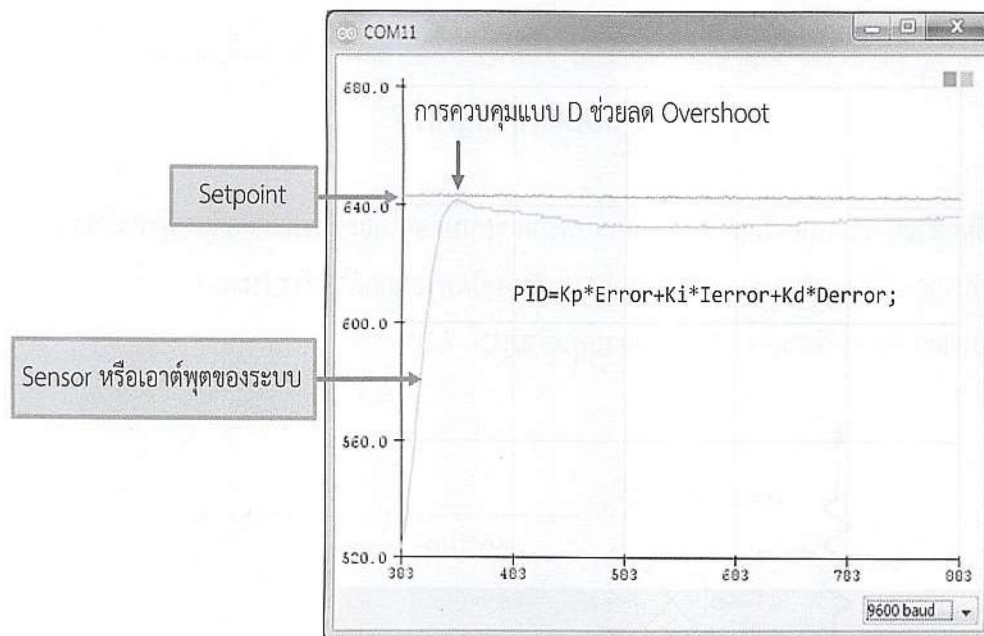
ตัวอย่างที่ 7.5 การควบคุมตำแหน่งคานแบบ PID

```
float Setpoint,Sensor,Error;
float Error1,PreError,Ierror,Derror;
float Kp=1.5,Ki=0.005,Kd=0.0001;        //กำหนดค่าเกณฑ์
float P,I,D,PIDcontrol;
void setup()
{
  Serial.begin(9600);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
}
  void loop()
{
    Setpoint = analogRead(A1);          //อ่านค่า VR1 = Setpoint
    Sensor = analogRead(A2);            //อ่านค่า VR2 = Sensor
    Error = Setpoint - Sensor;          //หาค่า Error
    if (Error < 0)
      Error1 = Error * (-1);            //ให้ Error1 มีค่าเป็นบวก
    else
      Error1 = Error;
Ierror = Ierror + Error;                //หาค่า Error สะสม
```

```
Derror = Error - PreError;          //หาค่า Derror
PreError = Error;                   //เก็บค่า Error ก่อนหน้านั้น
P = Kp * Error1;                    //P Control
I = Ki * Ierror;                    //I Control
D = Kd * Derror;                    //D Control
if (I > 220) I = 220;
if (D < -20) D = -20;
PIDcontrol = P+I+D;                 //PID Control
if (PIDcontrol > 230)
  PIDcontrol = 230;
if (PIDcontrol < 0)
  PIDcontrol = 0;
analogWrite(10, PIDcontrol);        //ควบคุมความเร็วมอเตอร์ 1
analogWrite(11, PIDcontrol);        //ควบคุมความเร็วมอเตอร์ 2
if (Error > 0)                      //ถ้า Error > 0 ให้หมุนคานลง
{ digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  digitalWrite(12, LOW);
  digitalWrite(13, HIGH);
}
if (Error < 0)                      //ถ้า Error < 0 ให้หมุนคานขึ้น
{ digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  digitalWrite(12, HIGH);
  digitalWrite(13, LOW);
}
  Serial.print(Setpoint);           //แสดงค่า Setpoint ที่ตั้งไว้
  Serial.print("\t");
  Serial.println(Sensor);           //แสดงค่า Sensor ที่วัดได้
}
```

ผลการรันโปรแกรม

โปรแกรมจะหาค่าผิดพลาดของตำแหน่งคาน จากนั้นคำนวณหาค่าเอาต์พุตด้วยการควบคุมแบบ PID นำสัญญาณเอาต์พุตมารวมกันแล้วส่งสัญญาณไปควบคุมมอเตอร์ให้หมุนไปยังตำแหน่งที่ตั้งไว้ ซึ่งอาจมีค่าผิดพลาดและแกว่งเล็กน้อย ต้องปรับค่าเกณฑ์ Kp Ki และ Kd ระบบจึงจะมีความเสถียร



รูปที่ 7.19 กราฟผลตอบสนองของระบบควบคุมตำแหน่งแบบ PID

7.8 สรุป

ระบบควบคุม หมายถึง ระบบควบคุมกระบวนการให้ทำงานตามวัตถุประสงค์ที่ต้องการ ซึ่งระบบควบคุมที่ดีควรมีเสถียรภาพ สามารถควบคุมสัญญาณเอาต์พุตให้เป็นไปตามค่าที่ตั้งไว้ มีการตอบสนองเร็วทันเวลา และมีความผิดพลาดน้อย

ค่าผิดพลาด คือ ค่าความแตกต่างระหว่างค่าที่ตั้งไว้กับผลตอบสนองของเอาต์พุต (Setpoint – Error) ปัญหาของระบบควบคุมคือจะทำอย่างไรให้ระบบมีความผิดพลาดน้อยที่สุด

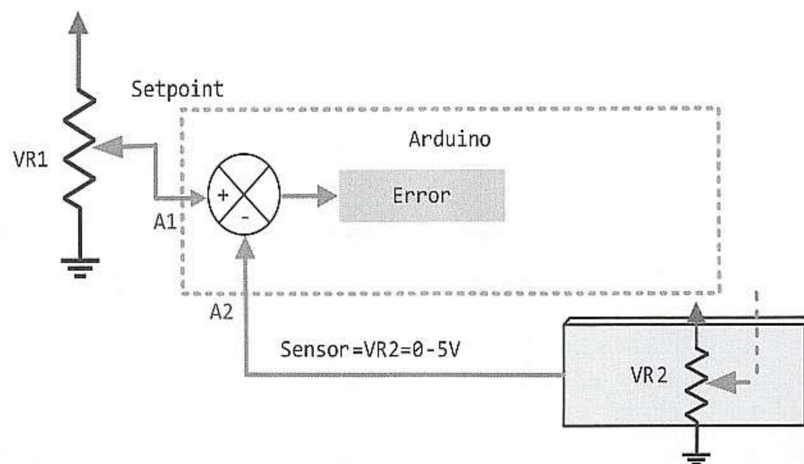
ระบบควบคุมแบบ PID เป็นระบบควบคุมแบบป้อนกลับที่นำเอาค่าผิดพลาดไปคำนวณหาค่าเอาต์พุตเพื่อควบคุมการทำงานของระบบ การควบคุมแบบ PID ประกอบไปด้วย การควบคุมแบบ P การควบคุมแบบ I และการควบคุมแบบ D ซึ่งสามารถคำนวณหาค่าต่าง ๆ ในระบบได้ดังนี้

```
Error=Setpoint-Sensor;
Ierror=Ierror+Error;
Derror=Error-PreError;
PreError=Error;
P=Kp*Error;
I=Ki*Ierror;
D=Kd*Derror;
PID=P+I+D;
```

การควบคุมแบบ PID จะทำให้ระบบควบคุมสามารถบังคับให้กระบวนการหรือผลตอบสนองของระบบมีเสถียรภาพ มีค่าผิดพลาดน้อยลง ผลของการควบคุมตำแหน่งแบบ PID ของคานคือทำให้คานหมุนไปยังตำแหน่งที่กำหนดโดยมีค่าผิดพลาดเพียงเล็กน้อย

แบบฝึกหัดบทที่ 7

1. ให้ยกตัวอย่างระบบควบคุมแบบป้อนกลับพร้อมเขียนบล็อกไดอะแกรมและอธิบายการทำงาน
2. อธิบายการทำงานของระบบควบคุมแบบแบบเปิด-ปิดพร้อมยกตัวอย่างประกอบ
3. อธิบายการหาค่าผิดพลาดของระบบควบคุมตามรูปที่ 7.20



รูปที่ 7.20 การหาค่าผิดพลาดของระบบควบคุม

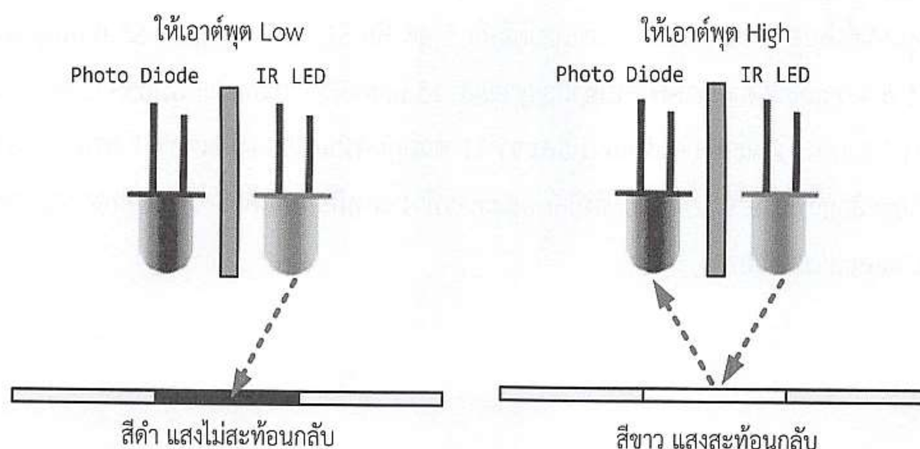
4. จากรูปที่ 7.20 Setpoint Sensor และ Error มีค่าอยู่ในช่วงใด
5. อธิบายการทำงานของการควบคุมแบบ P และเขียนสมการคำนวณหาค่าเอาต์พุตของการควบคุมแบบ P
6. อธิบายการทำงานของการควบคุมแบบ I และเขียนสมการคำนวณหาค่าเอาต์พุตของการควบคุมแบบ I
7. อธิบายการทำงานของการควบคุมแบบ D และเขียนสมการคำนวณหาค่าเอาต์พุตของการควบคุมแบบ D
8. เพราะเหตุใดการควบคุมแบบ P จึงเกิดค่าผิดพลาดในระยะคงตัว

บทที่ 8 หุ่นยนต์เดินตามเส้นแบบ PID

หุ่นยนต์เดินตามเส้นจะใช้เซนเซอร์ในการตรวจจับเส้นสีดำบนพื้นสีขาว แล้วส่งสัญญาณให้ไมโครคอนโทรลเลอร์ทำหน้าที่ตัดสินใจควบคุมมอเตอร์ให้หมุนซ้ายหรือหมุนขวาด้วยความเร็วค่าหนึ่ง เพื่อให้หุ่นยนต์เดินตรงไปตามเส้นทางที่กำหนด ในบทนี้จะกล่าวถึงหุ่นยนต์เดินตามเส้น วงจรควบคุม เซนเซอร์ตรวจจับเส้น การเขียนโปรแกรมควบคุมหุ่นยนต์ให้เดินตามเส้นแบบมีเงื่อนไข และการควบคุมหุ่นยนต์เดินตามเส้นแบบ PID

8.1 เซนเซอร์ตรวจจับเส้น

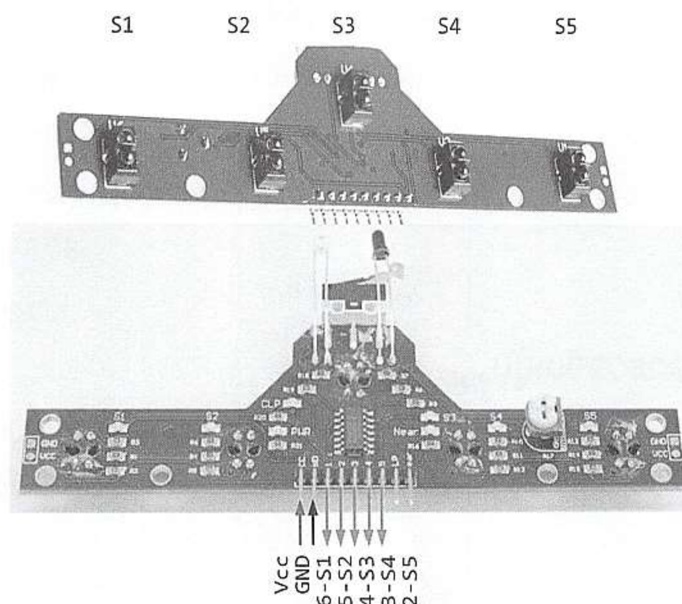
เซนเซอร์ตรวจจับเส้นสีดำและสีขาวเป็นอุปกรณ์ที่ใช้ในการตรวจจับการเปลี่ยนแปลงคุณสมบัติทางฟิสิกส์ของสีแล้วแปลงเป็นสัญญาณทางไฟฟ้า การตรวจจับเส้นทางจะใช้หลักการสะท้อนของแสงบนพื้นสีขาวกับสีดำ ตามคุณสมบัติทางฟิสิกส์ที่สีขาวจะสะท้อนแสงได้มากกว่าสีดำ จากนั้นแปลงแสงที่สะท้อนกลับเป็นสัญญาณไฟฟ้าส่งผ่านวงจรเปรียบเทียบเพื่อเปรียบเทียบสัญญาณที่รับได้ แล้วเปลี่ยนเป็นสัญญาณดิจิทัลเพื่อทำงานเชื่อมต่อกับไมโครคอนโทรลเลอร์ ในการทำงาน เมื่อตรวจจับเส้นสีดำจะไม่มีการสะท้อนกลับของสัญญาณ ให้เอาต์พุตเป็นลอจิก 0 เมื่อตรวจจับเส้นสีขาวจะมีการสะท้อนกลับของสัญญาณและให้เอาต์พุตเป็นลอจิก 1 หลักการทำงานแสดงดังรูปที่ 8.1



รูปที่ 8.1 หลักการทำงานของเซนเซอร์ตรวจจับเส้นสีดำและสีขาว

หุ่นยนต์เดินตามเส้นจะใช้เซนเซอร์ตรวจจับเส้นสีดำ 5 จุด (5 Channel Line Tracking Sensor Module หรือ BFD-1000) เป็นเซนเซอร์ที่มีความแม่นยำสูง สามารถตรวจจับเส้นสีดำบนพื้นขาวในระยะทาง 0–4 เซนติเมตร ให้สัญญาณเอาต์พุตแบบดิจิทัล มีไฟแสดงสถานะการทำงาน มีสวิตช์กันชนด้านหน้าและเซนเซอร์ตรวจจับสิ่งกีดขวางด้านหน้า สามารถปรับระยะทางได้ 0–5 เซนติเมตร ใช้แรงดันไฟ 3–5.5 โวลต์

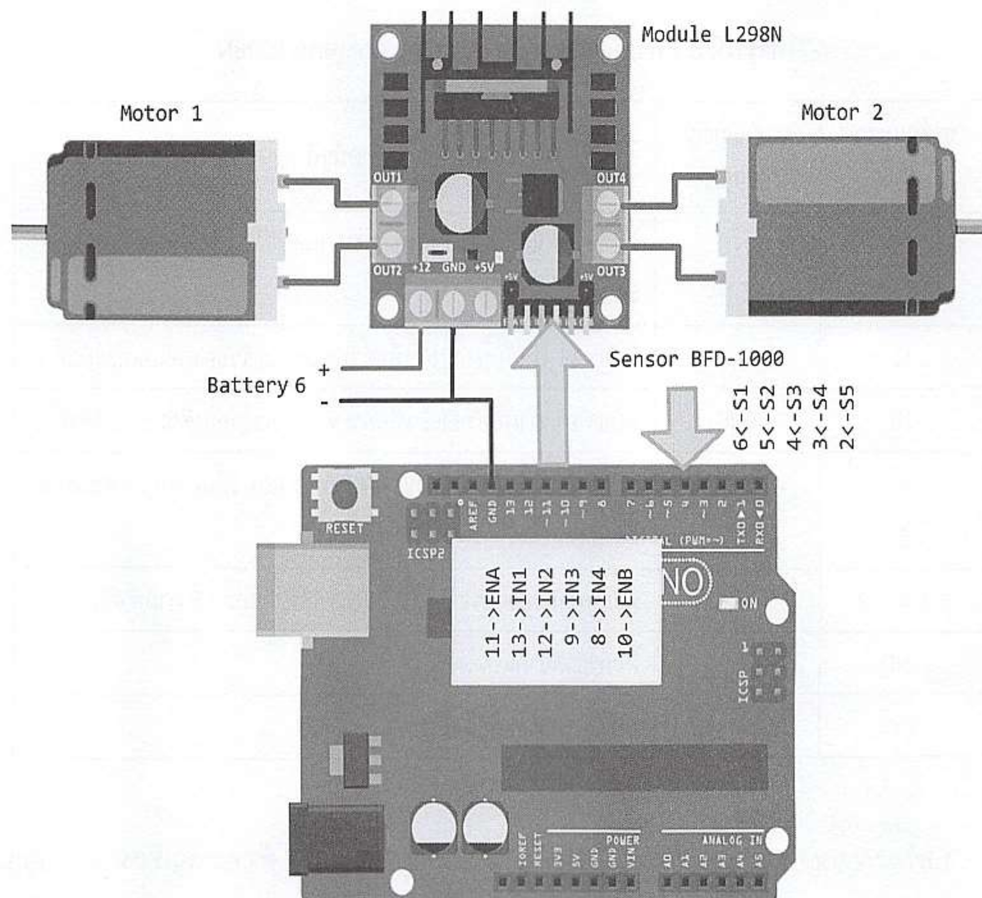
การทำงานของเซนเซอร์ เมื่อตรวจจับเจอเส้นสีดำจะให้เอาต์พุตเป็นลอจิก 0 หรือ LOW แต่จะมีสัญญาณไฟติดสว่างที่บอร์ดของเซนเซอร์ ถ้าตรวจจับเจอเส้นสีขาวจะให้เอาต์พุตเป็นลอจิก 1 หรือ HIGH และไม่มีสัญญาณไฟติดที่บอร์ดของเซนเซอร์



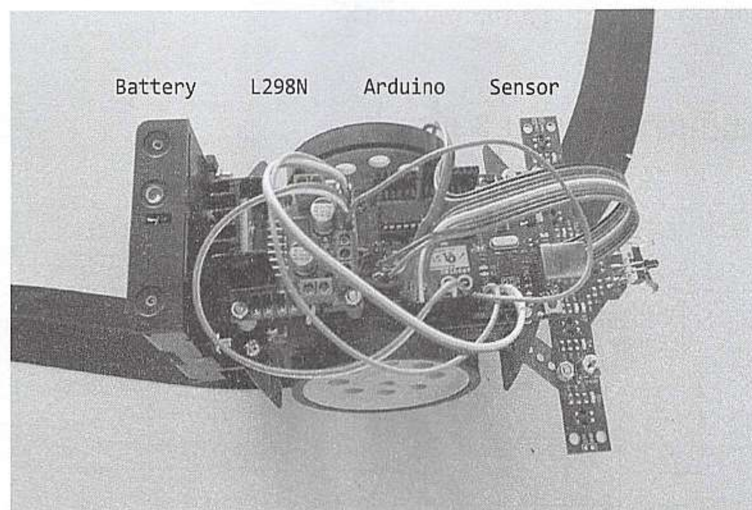
รูปที่ 8.2 เซนเซอร์ตรวจจับเส้นสีดำ 5 จุด (BFD-1000)

8.2 วงจรหุ่นยนต์เดินตามเส้น

หุ่นยนต์เดินตามเส้นจะใช้เซนเซอร์แบบดิจิทัล 5 จุด คือ S1 S2 S3 S4 และ S5 เชื่อมต่อเข้ากับขาสัญญาณ 6 5 4 3 และ 2 ตามลำดับ ส่วนขาสัญญาณ 8–13 และ GND จะต่อเข้ากับบอร์ด L298N สัญญาณเอาต์พุตขา 13 และ 12 จะควบคุมทิศทาง และขา 11 ควบคุมความเร็วของมอเตอร์ 1 ส่วนมอเตอร์ 2 จะควบคุมด้วยขาสัญญาณ 8 9 และ 10 ซึ่งมีลักษณะการทำงานเหมือนมอเตอร์ 1 การต่อขาสัญญาณแสดงดังรูปที่ 8.3 และตารางที่ 8.1



รูปที่ 8.3 วงจรควบคุมหุ่นยนต์เดินตามเส้น



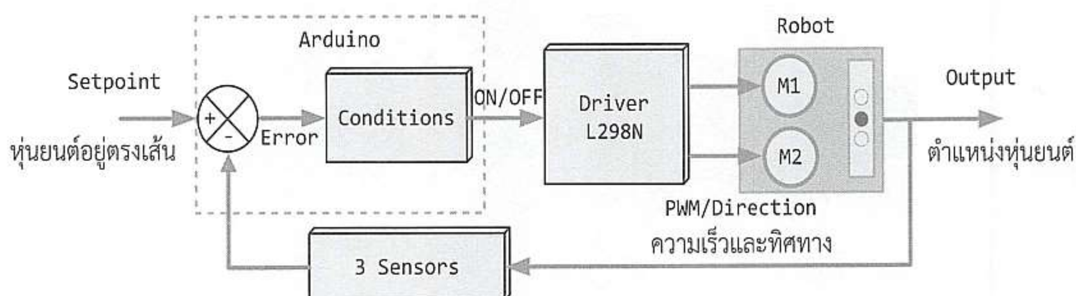
รูปที่ 8.4 หุ่นยนต์เดินตามเส้นและวงจรควบคุม

ตารางที่ 8.1 การต่อขาสัญญาณ Arduino และบอร์ด L298N

ขาสัญญาณ Arduino	ขาสัญญาณ L298N	หน้าที่
13 12	IN1 IN2	ต่อขา 13 และ 12 เข้ากับขา IN1 และ IN2 เพื่อควบคุมทิศทางของดีซีมอเตอร์ตัวที่ 1
11	ENA	ต่อขา 11 เข้ากับขา ENA เพื่อควบคุมความเร็วของดีซีมอเตอร์ตัวที่ 1
10	ENB	ต่อขา 10 เข้ากับขา ENB เพื่อควบคุมความเร็วของดีซีมอเตอร์ตัวที่ 2
9 8	IN3 IN4	ต่อขา 9 และ 8 เข้ากับขา IN3 และ IN4 เพื่อควบคุมทิศทางของดีซีมอเตอร์ตัวที่ 2
6 5 4 3 2		ต่อกับเซนเซอร์ตรวจจับเส้น S1 S2 S3 S4 และ S5 ตามลำดับ
GND	GND	ต่อกราวด์จากแบตเตอรี่
Vin	Vin 6–12	ต่อไฟจากแบตเตอรี่ 6 โวลต์

บอร์ด Motor Drive Module L298N สามารถขับดีซีมอเตอร์ได้ 2 ตัว ควบคุมทิศทางการหมุนของมอเตอร์ได้ที่ขาอินพุต IN1–IN4 และควบคุมความเร็วมอเตอร์แบบ PWM ที่ขา ENA และ ENB ใช้แรงดันไฟฟ้า 6–35 โวลต์ จ่ายกระแสสูงสุดข้างละ 2 แอมแปร์ มีแหล่งจ่ายไฟ 5 โวลต์ในบอร์ด การควบคุมทิศทางของดีซีมอเตอร์ตัวที่ 1 จะขึ้นอยู่กับขา IN1–IN2 ส่วนการควบคุมความเร็วในการหมุนของมอเตอร์จะขึ้นอยู่กับขา ENA ส่วนมอเตอร์ตัวที่ 2 ก็ทำงานในลักษณะเดียวกัน

8.3 หุ่นยนต์เดินตามเส้นแบบ 3 เงื่อนไข



รูปที่ 8.5 บล็อกไดอะแกรมการควบคุมหุ่นยนต์เดินตามเส้นแบบมีเงื่อนไข

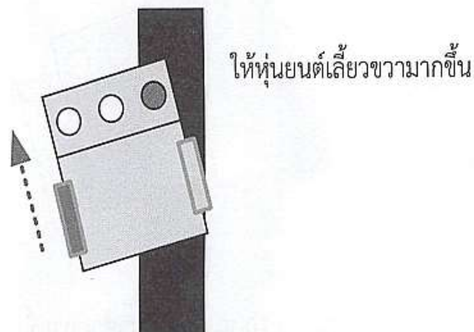
หุ่นยนต์เดินตามเส้นแบบ 3 เงื่อนไขจะตรวจวัดด้วยเซนเซอร์ 3 ตัว แล้วนำมากำหนดเงื่อนไขว่าจะให้หุ่นยนต์ตรงไป เลี้ยวซ้าย หรือเลี้ยวขวา โดยมีเงื่อนไขดังนี้

ถ้าค่าที่อ่านได้จากเซนเซอร์ทั้ง 3 ตัวเท่ากับ 1 0 0 หุ่นยนต์เบนซ้ายเล็กน้อยให้เลี้ยวขวาเล็กน้อย มอเตอร์ซ้ายหมุน มอเตอร์ขวาหยุดหมุน



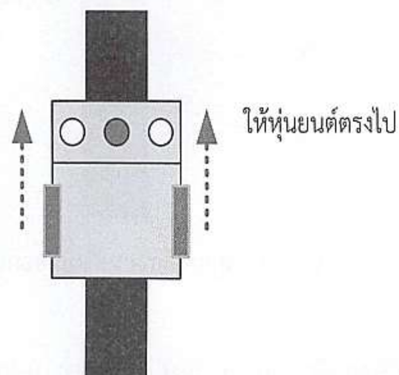
รูปที่ 8.6 หุ่นยนต์เลี้ยวขวาเล็กน้อย

ถ้าค่าที่อ่านได้จากเซนเซอร์ทั้ง 3 ตัวเท่ากับ 1 1 0 หุ่นยนต์เบนซ้ายให้เลี้ยวขวามากขึ้น มอเตอร์ซ้ายหมุน มอเตอร์ขวาหยุดหมุน



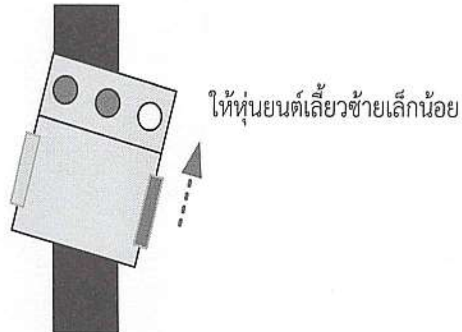
รูปที่ 8.7 หุ่นยนต์เลี้ยวขวามากขึ้น

ถ้าค่าที่อ่านได้จากเซนเซอร์ทั้ง 3 ตัวเท่ากับ 1 0 1 หุ่นยนต์อยู่ตรงเส้นให้ตรงไป มอเตอร์ซ้ายและมอเตอร์ขวาหมุนไปข้างหน้า



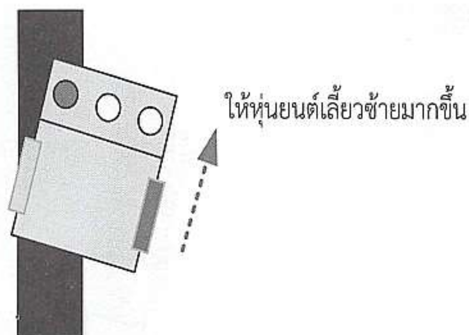
รูปที่ 8.8 หุ่นยนต์ตรงไป

ถ้าค่าที่อ่านได้จากเซนเซอร์ทั้ง 3 ตัวเท่ากับ 0 0 1 หุ่นยนต์เบนขวาเล็กน้อยให้เลี้ยวซ้ายเล็กน้อย มอเตอร์ซ้ายหยุดหมุน มอเตอร์ขวาหมุน



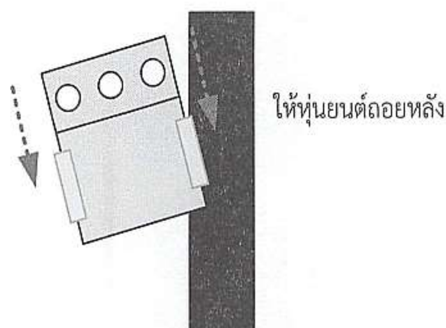
รูปที่ 8.9 หุ่นยนต์เลี้ยวซ้ายเล็กน้อย

ถ้าค่าที่อ่านได้จากเซนเซอร์ทั้ง 3 ตัวเท่ากับ 0 1 1 หุ่นยนต์เบนขวามากขึ้นให้เลี้ยวซ้ายมากขึ้น มอเตอร์ซ้ายหยุดหมุน มอเตอร์ขวาหมุน



รูปที่ 8.10 หุ่นยนต์เลี้ยวซ้ายมากขึ้น

ถ้าค่าที่อ่านได้จากเซนเซอร์ทั้ง 3 ตัวเท่ากับ 1 1 1 หุ่นยนต์อยู่นอกเส้นทางให้ถอยหลัง มอเตอร์ซ้ายและมอเตอร์ขวาหมุนกลับทาง



รูปที่ 8.11 หุ่นยนต์ถอยหลังเมื่อออกนอกเส้นทาง

การกำหนดเงื่อนไขอาจมีมากกว่านี้ เช่น การเจอทางแยกหรือเส้นทางขาด สามารถเขียนโปรแกรมทดสอบและควบคุมหุ่นยนต์เดินตามเส้นแบบมีเงื่อนไข ดังตัวอย่างที่ 8.1–8.5

ตัวอย่างที่ 8.1 โปรแกรมทดสอบหุ่นยนต์ให้ไปข้างหน้าและถอยกลับ

```
void setup()
{
    pinMode(8,OUTPUT);                 //ขา 8 ต่อกับมอเตอร์ 2
    pinMode(9,OUTPUT);                 //ขา 9 ต่อกับมอเตอร์ 2
    pinMode(10,OUTPUT);                //ขา 10 ต่อกับขา ENB
    pinMode(11,OUTPUT);                //ขา 11 ต่อกับขา ENA
    pinMode(12,OUTPUT);                //ขา 12 ต่อกับมอเตอร์ 1
    pinMode(13,OUTPUT);                //ขา 13 ต่อกับมอเตอร์ 1
}
void loop()
{
    int PWM_Speed=100;                 //กำหนดความเร็ว PWM = 100
    analogWrite(10,PWM_Speed);         //กำหนดความเร็วมอเตอร์ 2
    analogWrite(11,PWM_Speed);         //กำหนดความเร็วมอเตอร์ 1
    digitalWrite(8,LOW);               //ขา 8 = LOW
    digitalWrite(9,HIGH);              //ขา 9 = HIGH มอเตอร์ 2 หมุน
    digitalWrite(12,LOW);              //ขา 12 = LOW
    digitalWrite(13,HIGH);             //ขา 13 = HIGH มอเตอร์ 1 หมุน
    delay(1500);                       //หน่วงเวลา
    digitalWrite(9,LOW);               //ขา 9 = LOW มอเตอร์ 2 หยุดหมุน
    digitalWrite(13,LOW);              //ขา 13 = LOW มอเตอร์ 1 หยุดหมุน
    delay(2000);
    digitalWrite(8,HIGH);              //ขา 8 = HIGH มอเตอร์ 2 หมุนกลับทาง
    digitalWrite(12,HIGH);             //ขา 12 = HIGH มอเตอร์ 1 หมุนกลับทาง
    delay(1500);
}
```

ผลการรันโปรแกรม

หุ่นยนต์จะไปข้างหน้า 1.5 วินาที หยุด 2 วินาที แล้วถอยหลัง 1.5 วินาที หากหุ่นยนต์หมุนหรือทำงานสลับกันให้สลับขาของของมอเตอร์ และให้สังเกตแนวการเคลื่อนที่ของหุ่นยนต์ หากไม่ตรงให้ปรับที่โครงสร้างหรือล้อเพื่อให้หุ่นยนต์วิ่งได้ตรง

ตัวอย่างที่ 8.2 โปรแกรมทดสอบหุ่นยนต์ให้เลี้ยวซ้าย

```
void setup()
{
  pinMode(8,OUTPUT);            //ขา 8 ต่อกับมอเตอร์ 2
  pinMode(9,OUTPUT);            //ขา 9 ต่อกับมอเตอร์ 2
  pinMode(10,OUTPUT);           //ขา 10 ต่อกับขา ENB
  pinMode(11,OUTPUT);           //ขา 11 ต่อกับขา ENA
  pinMode(12,OUTPUT);           //ขา 12 ต่อกับมอเตอร์ 1
  pinMode(13,OUTPUT);           //ขา 13 ต่อกับมอเตอร์ 1
}
void loop()
{
  int PWM_Speed=120;            //กำหนดความเร็วของมอเตอร์
  analogWrite(10,PWM_Speed);    //กำหนดความเร็วมอเตอร์ 2
  analogWrite(11,PWM_Speed);    //กำหนดความเร็วมอเตอร์ 1
  digitalWrite(8,LOW);          //ขา 8 = LOW
  digitalWrite(9,HIGH);         //ขา 9 = HIGH มอเตอร์ 2 หมุน
  digitalWrite(12,LOW);         //ขา 12 = LOW มอเตอร์ 1 หยุดหมุน
  digitalWrite(13,LOW);         //ขา 13 = LOW มอเตอร์ 1 หยุดหมุน
  delay(1000);
  digitalWrite(9,LOW);          //ขา 9 = LOW มอเตอร์ 2 หยุดหมุน
  delay(2000);
}
```

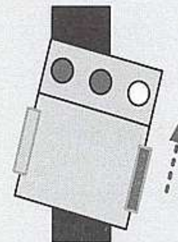
ผลการรันโปรแกรม

หุ่นยนต์จะเลี้ยวซ้าย 1 วินาที แล้วหยุด 2 วินาที หากหุ่นยนต์เลี้ยวผิดทางให้สลับขาของมอเตอร์ การปรับความเร็วของมอเตอร์สามารถปรับที่ตัวแปร PWM_Speed ซึ่งสามารถปรับความเร็วได้สูงสุดถึง 255 แต่การปรับความเร็วที่มากเกินไปอาจทำให้หุ่นยนต์วิ่งออกนอกเส้นทางหรือควบคุมได้ยาก

ตัวอย่างที่ 8.3 โปรแกรมควบคุมหุ่นยนต์ให้เดินตามเส้นแบบ 3 เงื่อนไข

```
void Forward();                  //ฟังก์ชันให้หุ่นยนต์เดินหน้า
void Turn_L();                   //ฟังก์ชันให้หุ่นยนต์เลี้ยวซ้าย
void Turn_R();                   //ฟังก์ชันให้หุ่นยนต์เลี้ยวขวา
void setup()
  {
    pinMode(8,OUTPUT);           //กำหนดสัญญาณเอาต์พุต
    pinMode(9,OUTPUT);
    pinMode(10,OUTPUT);
    pinMode(11,OUTPUT);
    pinMode(12,OUTPUT);
    pinMode(13,OUTPUT);
    pinMode(3,INPUT);            //กำหนดอินพุตเซนเซอร์เส้น S4
    pinMode(4,INPUT);            //กำหนดอินพุตเซนเซอร์เส้น S3
    pinMode(5,INPUT);            //กำหนดอินพุตเซนเซอร์เส้น S2
  }
void loop()
  {
  int PWM_Speed=120;             //กำหนดความเร็วของมอเตอร์
  char S2,S3,S4;
  S2=digitalRead(5);             //อ่านค่าเซนเซอร์จากขา 5
  S3=digitalRead(4);             //อ่านค่าเซนเซอร์จากขา 4
  S4=digitalRead(3);             //อ่านค่าเซนเซอร์จากขา 3
  analogWrite(10,PWM_Speed);     //กำหนดความเร็วมอเตอร์ 2
  analogWrite(11,PWM_Speed);     //กำหนดความเร็วมอเตอร์ 1
  if(S2==1&&S3==0&&S4==1)        //ถ้าหุ่นยนต์อยู่ตรงเส้น
    Forward();                   //ให้หุ่นยนต์เดินตรง
  if(S4==0)                      //ถ้าเบนซ้าย
    Turn_R();                    //ให้หุ่นยนต์เลี้ยวขวา
  if(S2==0)                      //ถ้าเบนขวา
    Turn_L();                    //ให้หุ่นยนต์เลี้ยวซ้าย
```

```
  delay(1);
  }
void Forward()                    //ฟังก์ชันให้หุ่นยนต์เดินหน้า
  {                               //ให้มอเตอร์ทั้งสองตัวเดินหน้า
    digitalWrite(8,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(12,LOW);
    digitalWrite(13,HIGH);
  }
void Turn_R()                     //ฟังก์ชันให้หุ่นยนต์เลี้ยวขวา
  {                               //ให้มอเตอร์ซ้ายเดินหน้า
    digitalWrite(8,LOW);
    digitalWrite(9,LOW);
    digitalWrite(12,LOW);
    digitalWrite(13,HIGH);
  }
void Turn_L()                     //ฟังก์ชันให้หุ่นยนต์เลี้ยวซ้าย
  {                               //ให้มอเตอร์ขวาเดินหน้า
    digitalWrite(8,LOW);
    digitalWrite(9,HIGH);
    digitalWrite(12,LOW);
    digitalWrite(13,LOW);
  }
```



ผลการรันโปรแกรม

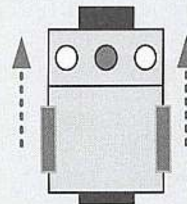
ไมโครคอนโทรลเลอร์รอรับสัญญาณจากเซนเซอร์แล้วส่งสัญญาณควบคุมให้หุ่นยนต์เดินไปตามเส้น โดยจะเลี้ยวซ้าย เลี้ยวขวา และตรงไป ตามเงื่อนไขของโปรแกรมดังนี้

- ถ้าเซนเซอร์ตัวกลางทำงาน S2 = 1 S3 = 0 S4 = 1 หุ่นยนต์จะตรงไป
- ถ้าเซนเซอร์ตัวขวาทำงาน S4 = 0 แสดงว่าหุ่นยนต์กำลังเบนออกไปทางซ้ายให้หุ่นยนต์เลี้ยวขวา
- ถ้าเซนเซอร์ตัวซ้ายทำงาน S2 = 0 แสดงว่าหุ่นยนต์กำลังเบนออกไปทางขวาให้หุ่นยนต์เลี้ยวซ้าย

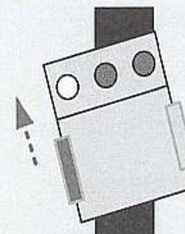
ตัวอย่างที่ 8.4 โปรแกรมควบคุมหุ่นยนต์ให้เดินตามเส้นแบบ 5 เงื่อนไข

```
void Forward();                         //ฟังก์ชันให้หุ่นยนต์เดินหน้า
void Turn_L();                          //ฟังก์ชันให้หุ่นยนต์เลี้ยวซ้าย
void Turn_R();                          //ฟังก์ชันให้หุ่นยนต์เลี้ยวขวา
void setup()
  {
    pinMode(8,OUTPUT);                  //กำหนดขา 8-13 ให้เป็นเอาต์พุต
    pinMode(9,OUTPUT);
    pinMode(10,OUTPUT);
    pinMode(11,OUTPUT);
    pinMode(12,OUTPUT);
    pinMode(13,OUTPUT);
    pinMode(3,INPUT);    //S4           //กำหนดขา 3 4 5 ให้เป็นอินพุต
    pinMode(4,INPUT);    //S3
    pinMode(5,INPUT);    //S2
  }
void loop()
  {
  int PWM_Speed=120;                    //กำหนดความเร็วของมอเตอร์
  char S2,S3,S4;
  S4=digitalRead(3);                    //อ่านค่าเซนเซอร์จากขา 3
  S3=digitalRead(4);                    //อ่านค่าเซนเซอร์จากขา 4
  S2=digitalRead(5);                    //อ่านค่าเซนเซอร์จากขา 5
  analogWrite(10,PWM_Speed);
  analogWrite(11,PWM_Speed);
    if(S2==1&&S3==0&&S4==1)             //ถ้าหุ่นยนต์อยู่ตรงเส้น
      Forward();                        //ให้วิ่งตรงไป
    if(S2==1&&S3==0&&S4==0)             //ถ้าหุ่นยนต์เบนซ้ายเล็กน้อย
      Turn_R();                         //ให้หุ่นยนต์เลี้ยวขวาเล็กน้อย
    if(S2==1&&S3==1&&S4==0)             //ถ้าหุ่นยนต์เบนซ้ายมาก
    { Turn_R();                         //ให้หุ่นยนต์เลี้ยวขวามากขึ้น
```

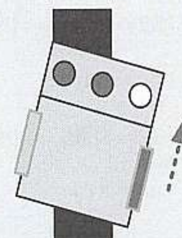
```
    delay(2);
   }
  if(S2==0&&S3==0&&S4==1)          //ถ้าหุ่นยนต์เบนขวาเล็กน้อย
    Turn_L();                      //ให้หุ่นยนต์เลี้ยวซ้ายเล็กน้อย
  if(S2==0&&S3==1&&S4==1)          //ถ้าหุ่นยนต์เบนขวามาก
   { Turn_L();                     //ให้หุ่นยนต์เลี้ยวซ้ายมากขึ้น
     delay(2);
   }
   delay(1);
   }
   void Forward()                  //ฟังก์ชันให้หุ่นยนต์เดินหน้า
   {                               //ให้มอเตอร์ทั้งสองตัวเดินหน้า
     digitalWrite(8,LOW);
     digitalWrite(9,HIGH);
     digitalWrite(12,LOW);
     digitalWrite(13,HIGH);
   }
```



```
 void Turn_R()                     //ฟังก์ชันให้หุ่นยนต์เลี้ยวขวา
   {                               //ให้มอเตอร์ซ้ายเดินหน้า
     digitalWrite(8,LOW);
     digitalWrite(9,LOW);
     digitalWrite(12,LOW);
     digitalWrite(13,HIGH);
   }
```



```
 void Turn_L()                     //ฟังก์ชันให้หุ่นยนต์เลี้ยวซ้าย
   {                               //ให้มอเตอร์ขวาเดินหน้า
     digitalWrite(8,LOW);
     digitalWrite(9,HIGH);
     digitalWrite(12,LOW);
     digitalWrite(13,LOW);
   }
```



ผลการรันโปรแกรม

ไมโครคอนโทรลเลอร์รอรับสัญญาณจากเซนเซอร์แล้วส่งสัญญาณควบคุมให้หุ่นยนต์เดินทางตามเส้น โดยจะเลี้ยวซ้าย เลี้ยวขวา และเดินตรงไป ตามเงื่อนไขดังนี้

- ถ้าเซนเซอร์ตัวกลางทำงาน S2 = 1 S3 = 0 S4 = 1 หุ่นยนต์จะตรงไป
- ถ้าเซนเซอร์ตัวกลางและตัวขวาทำงาน S2 = 1 S3 = 0 S4 = 0 แสดงว่าหุ่นยนต์กำลังเบนไปทางซ้ายเล็กน้อย ให้หุ่นยนต์เลี้ยวขวาเล็กน้อย
- ถ้าเซนเซอร์ตัวขวาทำงาน S2 = 1 S3 = 1 S4 = 0 แสดงว่าหุ่นยนต์กำลังเบนไปทางซ้ายมาก ให้หุ่นยนต์เลี้ยวขวามากขึ้น
- ถ้าเซนเซอร์ตัวกลางและตัวซ้ายทำงาน S2 = 0 S3 = 0 S4 = 1 แสดงว่าหุ่นยนต์กำลังเบนไปทางขวาเล็กน้อย ควบคุมให้หุ่นยนต์เลี้ยวซ้ายเล็กน้อย
- ถ้าเซนเซอร์ตัวซ้ายทำงาน S2 = 0 S3 = 1 S4 = 1 แสดงว่าหุ่นยนต์กำลังเบนไปทางขวามาก ควบคุมให้หุ่นยนต์เลี้ยวซ้ายมากขึ้น

ตัวอย่างที่ 8.5 โปรแกรมควบคุมหุ่นยนต์ให้เดินตามเส้นแบบปรับความเร็วของมอเตอร์

```
int SpeedA,SpeedB;                  //ประกาศตัวแปร
void setup()
{ pinMode(8,OUTPUT);                //กำหนดขาสัญญาณเอาต์พุต
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
  pinMode(3,INPUT);                 //กำหนดขาสัญญาณอินพุต S2-S4
  pinMode(4,INPUT);
  pinMode(5,INPUT);
}
void loop()
{ int PWM=130;                      //กำหนดความเร็วเริ่มต้นของมอเตอร์
  char S2,S3,S4;
  S2=digitalRead(5);                //อ่านค่าจากเซนเซอร์
  S3=digitalRead(4);
  S4=digitalRead(3);
  if(S2==1&&S3==0&&S4==1)           //ถ้าหุ่นยนต์อยู่ตรงเส้น
    { SpeedA=0;                     //ใช้ความเร็วปกติ PWM = 130
      SpeedB=0;
    }
    if(S2==1&&S3==0&&S4==0)         //ถ้าหุ่นยนต์เบนซ้ายเล็กน้อย
      SpeedA=SpeedA+1;              //ให้เพิ่มค่าตัวแปร SpeedA
    if(S2==1&&S3==1&&S4==0)         //ถ้าหุ่นยนต์เบนซ้ายมาก
      SpeedA=SpeedA+2;              //ให้เพิ่มค่าตัวแปร SpeedA ทีละ 2
    if(S2==0&&S3==0&&S4==1)         //ถ้าหุ่นยนต์เบนขวาเล็กน้อย
      SpeedB=SpeedB+1;              //ให้เพิ่มค่าตัวแปร SpeedB
    if(S2==0&&S3==1&&S4==1)         //ถ้าหุ่นยนต์เบนขวามาก
      SpeedB=SpeedB+2;              //ให้เพิ่มค่าตัวแปร SpeedB ทีละ 2
```

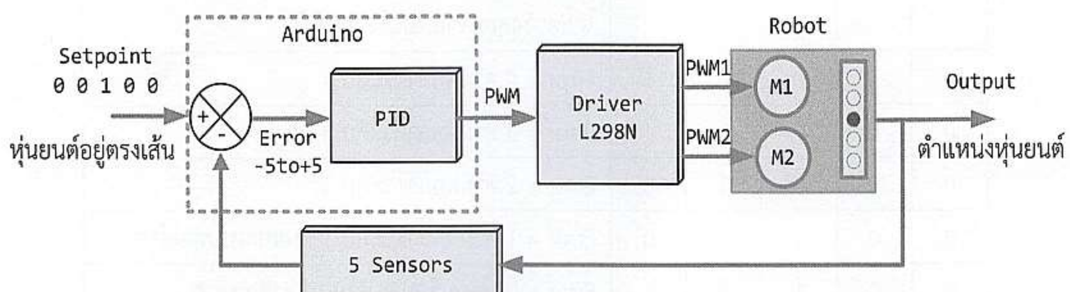
```
31    if(SpeedA>120) SpeedA=120;          //จำกัดความเร็ว
32    if(SpeedB>120) SpeedB=120;
33    analogWrite(10,PWM-SpeedA);         //ลดความเร็วของมอเตอร์ขวา
34    analogWrite(11,PWM-SpeedB);         //ลดความเร็วของมอเตอร์ซ้าย
35    digitalWrite(8,LOW);                //35-38 ให้หุ่นยนต์ตรงไป
36    digitalWrite(9,HIGH);
37    digitalWrite(12,LOW);
38    digitalWrite(13,HIGH);
39    delay(1);
40  }
```

ผลการรันโปรแกรม

เมื่อหุ่นยนต์อยู่ตรงเส้น ความเร็วของมอเตอร์ทั้งด้านซ้ายและขวาจะเท่ากัน PWM = 130 หากหุ่นยนต์อยู่นอกเส้นทาง ความเร็วของมอเตอร์ด้านใดด้านหนึ่งจะลดลง เพื่อควบคุมให้หุ่นยนต์กลับมาอยู่ในเส้นทาง

8.4 หุ่นยนต์เดินตามเส้นแบบ PID

การทำงานของหุ่นยนต์เดินตามเส้นแบบ PID โปรแกรมจะตรวจสอบตำแหน่งของหุ่นยนต์จากค่าเซนเซอร์ 5 จุด จากนั้นกำหนดค่าผิดพลาดจากเงื่อนไขของเซนเซอร์ มีค่าระหว่าง –5 ถึง +5 แล้วนำค่าผิดพลาดมาคำนวณตามสมการ PID เพื่อหาค่าแรงดัน PWM ที่เหมาะสมเพื่อควบคุมมอเตอร์ทั้งสองตัวให้หุ่นยนต์เดินตามเส้นแบบลื่นไหลไม่แกว่งไปมา



รูปที่ 8.12 บล็อกไดอะแกรมการควบคุมหุ่นยนต์เดินตามเส้นแบบ PID

Setpoint คือ ตำแหน่งที่ต้องการให้หุ่นยนต์อยู่ในเส้น S5 S4 S3 S2 S1 = 0 0 1 0 0

Sensor คือ เซนเซอร์ตรวจจับเส้นซึ่งเป็นเซนเซอร์แบบดิจิทัล 5 จุด

L298N คือ บอร์ดขยายสัญญาณเพื่อควบคุมความเร็วและทิศทางของมอเตอร์

PID คือ ระบบควบคุมที่นำค่าผิดพลาดมาคำนวณตามสมการ PID แล้วส่งสัญญาณไปควบคุมความเร็วของมอเตอร์ทั้งสองตัวให้หมุนด้วยความเร็วที่เหมาะสมเพื่อให้หุ่นยนต์วิ่งตรงเส้นทาง สามารถเขียนสมการได้ดังนี้

```
Error=Setpoint-Sensor;
Ierror=Ierror+Error;
Derror=Error-PreError;
PreError=Error;
P=Kp*Error;
I=Ki*Ierror;
D=Kd*Derror;
PID=P+I+D;
```

8.5 การหาค่าผิดพลาด

Error คือ ค่าผิดพลาดของตำแหน่งหุ่นยนต์ว่าออกนอกเส้นทางมากน้อยเพียงใด ซึ่งจะต้องทำการแปลงค่าผิดพลาดที่เป็นสัญญาณดิจิทัลให้เป็นค่าตัวเลข –5 ถึง +5 ดังตารางที่ 8.2

ตารางที่ 8.2 ระดับค่าผิดพลาดของเซนเซอร์ 5 จุด

เซนเซอร์					ค่าผิดพลาด
S1	S2	S3	S4	S5	
0	0	0	0	0	Error = 5 ค่าผิดพลาดระดับ 5 เบนซ้ายมากที่สุด โดยตรวจสอบจากเงื่อนไขก่อนหน้า
0	0	0	0	1	Error = 4 ค่าผิดพลาดระดับ 4
0	0	0	1	1	Error = 3 ค่าผิดพลาดระดับ 3
0	0	0	1	0	Error = 2 ค่าผิดพลาดระดับ 2
0	0	1	1	0	Error = 1 ค่าผิดพลาดระดับ 1 หุ่นยนต์เบนทางซ้าย
0	0	1	0	0	Error = 0 หุ่นยนต์อยู่ตรงเส้นไม่มีค่าผิดพลาด
0	1	1	0	0	Error = –1 ค่าผิดพลาดระดับ –1 หุ่นยนต์เบนทางขวา
0	1	0	0	0	Error = –2 ค่าผิดพลาดระดับ –2
1	1	0	0	0	Error = –3 ค่าผิดพลาดระดับ –3
1	0	0	0	0	Error = –4 ค่าผิดพลาดระดับ –4
0	0	0	0	0	Error = –5 ค่าผิดพลาดระดับ –5 เบนขวามากที่สุด

กรณีเงื่อนไข S1 ถึง S5 เท่ากับ 1 หุ่นยนต์ออกนอกเส้นทางตรวจจับเส้นสีดำไม่ได้ ค่าผิดพลาดจะมีค่า Error = 5 หรือ –5 ขึ้นอยู่กับค่าผิดพลาดก่อนหน้านั้น คือถ้า Error = –4 ค่าผิดพลาดจะเท่ากับ –5 แต่ถ้าไม่ใช่ ค่าผิดพลาดจะเท่ากับ 5 การหาค่าผิดพลาดมาจากเซนเซอร์ 5 จุดแล้วนำมากำหนดเงื่อนไข จากตารางที่ 8.2 สามารถเขียนโปรแกรมได้ดังนี้

```
if(S1==1&&S2==1&&S3==1&&S4==1&&S5==0)
   Error=4;
if(S1==1&&S2==1&&S3==1&&S4==0&&S5==0)
   Error=3;
if(S1==1&&S2==1&&S3==1&&S4==0&&S5==1)
   Error=2;
if(S1==1&&S2==1&&S3==0&&S4==0&&S5==1)
   Error=1;
if(S1==1&&S2==1&&S3==0&&S4==1&&S5==1)
   Error=0;
if(S1==1&&S2==0&&S3==0&&S4==1&&S5==1)
   Error=-1;
if(S1==1&&S2==0&&S3==1&&S4==1&&S5==1)
   Error=-2;
if(S1==0&&S2==0&&S3==1&&S4==1&&S5==1)
   Error=-3;
if(S1==0&&S2==1&&S3==1&&S4==1&&S5==1)
   Error=-4;
if(S1==1&&S2==1&&S3==1&&S4==1&&S5==1)
   { if(Error<0) Error=-5;
     else Error=5;
   }
```

ตัวอย่างที่ 8.6 โปรแกรมควบคุมหุ่นยนต์ให้เดินตามเส้นแบบ P

```
float Kp=12;
float P_Speed,Error;
void setup()
{
   pinMode(8,OUTPUT);
   pinMode(9,OUTPUT);
   pinMode(10,OUTPUT);
   pinMode(11,OUTPUT);
   pinMode(12,OUTPUT);
   pinMode(13,OUTPUT);
   pinMode(2,INPUT);   //S1
   pinMode(3,INPUT);   //S2
   pinMode(4,INPUT);   //S3
   pinMode(5,INPUT);   //S4
   pinMode(6,INPUT);   //S5
}
void loop()
{
   int PWM_Speed=120;
   char S1,S2,S3,S4,S5;
   S1=digitalRead(2);
   S2=digitalRead(3);
   S3=digitalRead(4);
   S4=digitalRead(5);
   S5=digitalRead(6);
   if(S1==1&&S2==1&&S3==1&&S4==1&&S5==0)
      Error=4;
   if(S1==1&&S2==1&&S3==1&&S4==0&&S5==0)
      Error=3;
   if(S1==1&&S2==1&&S3==1&&S4==0&&S5==1)
```

```
      Error=2;
    if(S1==1&&S2==1&&S3==0&&S4==0&&S5==1)
      Error=1;
    if(S1==1&&S2==1&&S3==0&&S4==1&&S5==1)
    {  Error=0;
       Ierror=0;
       Derror=0;
    }
  if(S1==1&&S2==0&&S3==0&&S4==1&&S5==1)
    Error=-1;
  if(S1==1&&S2==0&&S3==1&&S4==1&&S5==1)
    Error=-2;
  if(S1==0&&S2==0&&S3==1&&S4==1&&S5==1)
    Error=-3;
  if(S1==0&&S2==1&&S3==1&&S4==1&&S5==1)
    Error=-4;
  if(S1==1&&S2==1&&S3==1&&S4==1&&S5==1)
    {  if(Error<0)
          Error=-5;
       else
          Error=5;
    }
  P_Speed=Kp*Error;       //P Control
  if(P_Speed >100) P_Speed=100;
  if(P_Speed<-100) P_Speed=-100;
  analogWrite(10,PWM_Speed-P_Speed);
  analogWrite(11,PWM_Speed+P_Speed);
  digitalWrite(8,LOW);
  digitalWrite(9,HIGH);
  digitalWrite(12,LOW);
  digitalWrite(13,HIGH);
  }
```

ผลการรันโปรแกรม

ไมโครคอนโทรลเลอร์จะคำนวณหาค่าผิดพลาดจากเซนเซอร์ที่อ่านได้ ซึ่งมีค่าระหว่าง –5 ถึง +5 จากนั้นนำมาคำนวณตามสมการ P แล้วส่งสัญญาณไปควบคุมความเร็วของมอเตอร์ทั้งสองข้างเพื่อให้หุ่นยนต์เดินตามเส้นอย่างลื่นไหล ในการควบคุม ให้ทดลองปรับค่า Kp ให้เหมาะสมเพื่อไม่ให้หุ่นยนต์หลุดออกนอกเส้นทาง

```
P_Speed=Kp*Error; //-60 to 60
analogWrite(10,PWM_Speed-P_Speed); //60 to 180
analogWrite(11,PWM_Speed+P_Speed); //60 to 180
```

จากโปรแกรมตัวอย่าง กำหนดค่า Kp = 12 ทำให้ค่า P_Speed อยู่ในช่วง 12 x (–5) = –60 ถึง 12 x 5 = 60 ส่วนค่าสัญญาณ PWM ที่ส่งออกไปควบคุมมอเตอร์จะอยู่ในช่วง 120 – 60 ถึง 120 + 60

ค่าที่คำนวณได้จากสมการ PID จะส่งออกไปควบคุมมอเตอร์ทั้งสองข้างเพื่อควบคุมความเร็วให้เหมาะสม ซึ่งสัญญาณที่ส่งออกจะเป็นสัญญาณ PWM ที่มีค่าอยู่ระหว่าง 0–255 โดยค่า 0 หมายถึงมอเตอร์หยุดหมุน ส่วน 255 หมายถึงมอเตอร์หมุนด้วยความเร็วสูงสุด ในโปรแกรมตัวอย่างนี้จะใช้ค่าสูงสุดประมาณ 180–200 เพื่อไม่ให้หุ่นยนต์วิ่งเร็วเกินไปจนไม่สามารถควบคุมได้

ตัวอย่างที่ 8.7 โปรแกรมควบคุมหุ่นยนต์ให้เดินตามเส้นแบบ PID

```
float Kp=15,Ki=0.02,Kd=5,PID_Speed;
int Error,Ierror,Derror,Pre_Error;
void setup()
{
   pinMode(8,OUTPUT);
   pinMode(9,OUTPUT);
   pinMode(10,OUTPUT);
   pinMode(11,OUTPUT);
   pinMode(12,OUTPUT);
   pinMode(13,OUTPUT);
   pinMode(2,INPUT);   //S1
   pinMode(3,INPUT);   //S2
   pinMode(4,INPUT);   //S3
   pinMode(5,INPUT);   //S4
```

```
   pinMode(6,INPUT);   //S5
}
void loop()
{  int PWM_Speed=140;
   char S1,S2,S3,S4,S5;
   S1=digitalRead(2);
   S2=digitalRead(3);
   S3=digitalRead(4);
   S4=digitalRead(5);
   S5=digitalRead(6);
   if(S1==1&&S2==1&&S3==1&&S4==1&&S5==0)
      Error=4;
   if(S1==1&&S2==1&&S3==1&&S4==0&&S5==0)
      Error=3;
   if(S1==1&&S2==1&&S3==1&&S4==0&&S5==1)
      Error=2;
   if(S1==1&&S2==1&&S3==0&&S4==0&&S5==1)
      Error=1;
   if(S1==1&&S2==1&&S3==0&&S4==1&&S5==1)
   {  Error=0;
      Ierror=0;
      Derror=0;
   }
if(S1==1&&S2==0&&S3==0&&S4==1&&S5==1)
   Error=-1;
if(S1==1&&S2==0&&S3==1&&S4==1&&S5==1)
   Error=-2;
if(S1==0&&S2==0&&S3==1&&S4==1&&S5==1)
   Error=-3;
if(S1==0&&S2==1&&S3==1&&S4==1&&S5==1)
   Error=-4;
if(S1==1&&S2==1&&S3==1&&S4==1&&S5==1)
```

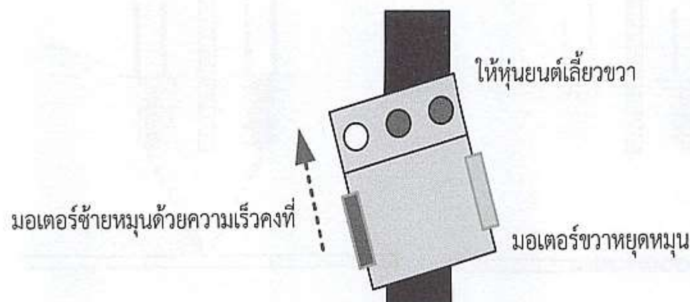
```
{  if(Error<0)
       Error=-5;
   else
       Error=5;
}
Ierror=Ierror+Error;
Derror=Error-Pre_Error;
PID_Speed=Kp*Error+Ki*Ierror+Kd*Derror; //PID control
Pre_Error=Error;
if(PID_Speed >100) PID_Speed=100;
if(PID_Speed<-100) PID_Speed=-100;
analogWrite(10,PWM_Speed-PID_Speed);
analogWrite(11,PWM_Speed+PID_Speed);
digitalWrite(8,LOW);
digitalWrite(9,HIGH);
digitalWrite(12,LOW);
digitalWrite(13,HIGH);
}
```

ผลการรันโปรแกรม

การควบคุมแบบ PID จะทำให้หุ่นยนต์วิ่งแบบลื่นไหลโดยไม่กระตุก การทำงานเริ่มจากไมโคร-คอนโทรลเลอร์จะคำนวณหาค่าผิดพลาดจากเซนเซอร์ นำมาคำนวณตามสมการ PID แล้วส่งสัญญาณไปควบคุมความเร็วของมอเตอร์ทั้งสองข้างเพื่อให้หุ่นยนต์เดินตามเส้น การควบคุมให้ทดลองปรับค่า Kp Ki และ Kd ทีละตัวแปรให้เหมาะสม เพื่อให้หุ่นยนต์เดินตามเส้นได้ลื่นไหลไม่ส่ายไปมา

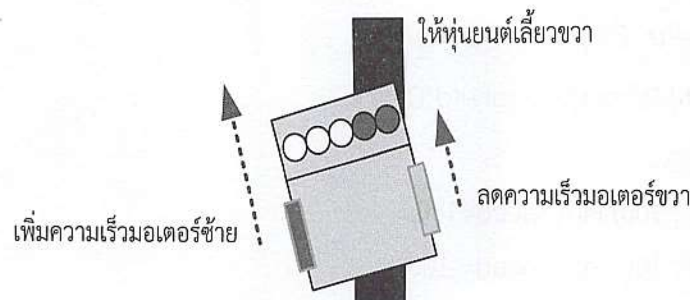
8.6 สรุป

การควบคุมหุ่นยนต์เดินตามเส้นแบบมีเงื่อนไขจะตรวจจับสัญญาณจากเซนเซอร์ 3 ตัวแล้วนำมากำหนดเงื่อนไขว่าจะให้หุ่นยนต์ตรงไป เลี้ยวซ้าย หรือเลี้ยวขวา โดยการควบคุมมอเตอร์ด้านซ้ายและด้านขวาให้ทำงานหรือหยุดทำงานด้วยความเร็วคงที่ซึ่งอาจทำให้หุ่นยนต์เดินส่ายเล็กน้อย เมื่อหุ่นยนต์เบนไปทางซ้าย มอเตอร์ขวาจะหยุด แล้วมอเตอร์ซ้ายหมุนไปหน้าทำให้หุ่นยนต์เลี้ยวขวา



รูปที่ 8.13 การควบคุมแบบมีเงื่อนไขเมื่อหุ่นยนต์เบนไปทางซ้าย

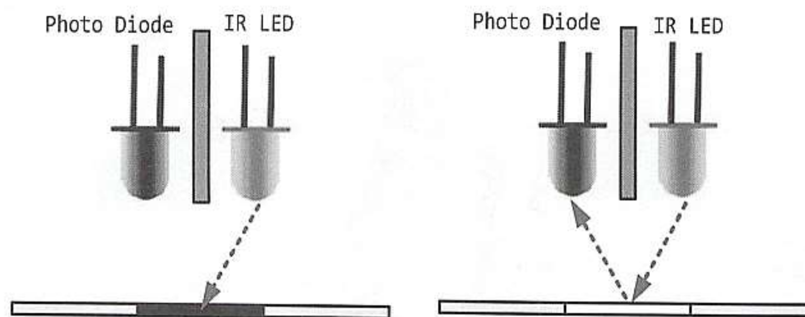
การควบคุมหุ่นยนต์เดินตามเส้นแบบ PID ไมโครคอนโทรลเลอร์จะตรวจสอบตำแหน่งของหุ่นยนต์จากเซนเซอร์ทั้ง 5 จุด จากนั้นหาค่าผิดพลาดซึ่งมีค่าระหว่าง –5 ถึง +5 แล้วนำมาคำนวณตามสมการ PID เพื่อหาค่าแรงดัน PWM ที่เหมาะสมเพื่อไปควบคุมความเร็วของมอเตอร์ทั้งสองตัวให้หุ่นยนต์เดินตามเส้นแบบลื่นไหลไม่แกว่งไปมา เมื่อหุ่นยนต์เบนไปทางซ้าย มอเตอร์ขวาจะลดความเร็วและมอเตอร์ซ้ายจะเพิ่มความเร็วทำให้หุ่นยนต์เลี้ยวขวา



รูปที่ 8.14 การควบคุมแบบ PID เมื่อหุ่นยนต์เบนไปทางซ้าย

แบบฝึกหัดบทที่ 8

1. อธิบายการทำงานของเซนเซอร์ในการตรวจจับเส้นดังรูปที่ 8.15

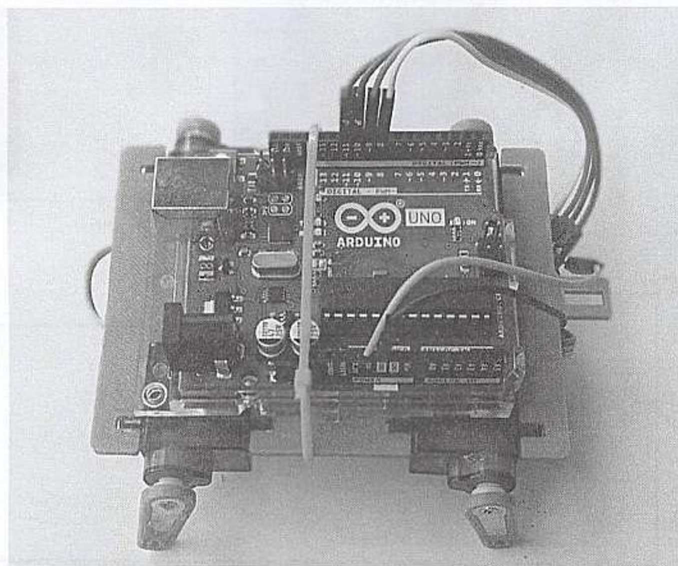


รูปที่ 8.15 การทำงานของเซนเซอร์

2. อธิบายหลักการควบคุมความเร็วของมอเตอร์แบบ PWM
3. อธิบายคำสั่ง
   - if(S2==1&&S3==0&&S4==1)
4. อธิบายหลักการหาค่าผิดพลาดของระบบควบคุมแบบ PID
5. อธิบายคำสั่งต่อไปนี้
   - Ierror=Ierror+Error;
   - Derror=Error-Pre_Error;
   - PID_Speed=Kp*Error+Ki*Ierror+Kd*Derror;
   - Pre_Error=Error;
   - if(PID_Speed >100) PID_Speed=100;
   - if(PID_Speed<-100) PID_Speed=-100;
   - analogWrite(10,PWM_Speed-PID_Speed);
   - analogWrite(11,PWM_Speed+PID_Speed);
6. ให้เขียนโปรแกรมควบคุมหุ่นยนต์ให้เดินตามเส้นวนเป็นวงกลม 3 รอบแล้วหยุด

บทที่ 9 หุ่นยนต์ 4 ขา

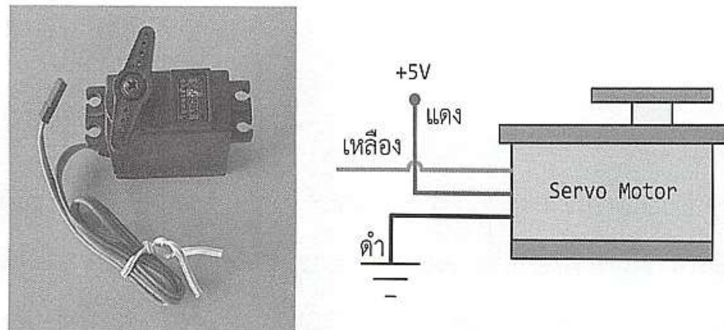
ในบทนี้จะกล่าวถึงการสร้างหุ่นยนต์ 4 ขา วงจรควบคุมการทำงาน หลักการเดินหน้า ถอยหลัง การเลี้ยวของหุ่นยนต์ 4 ขา การทำงานของเซอร์โวมอเตอร์ การเชื่อมต่อไมโครคอนโทรลเลอร์กับเซอร์โวมอเตอร์ การควบคุมตำแหน่งของเซอร์โวมอเตอร์ การเขียนโปรแกรมควบคุมการทำงานของเซอร์โวมอเตอร์ และการเขียนโปรแกรมควบคุมการเดินของหุ่นยนต์ ซึ่งหุ่นยนต์ 4 ขาที่กล่าวถึงนี้จะใช้เซอร์โวมอเตอร์ 4 ตัว ในการขับเคลื่อนแต่ละขา รูปที่ 9.1 แสดงโครงสร้างของหุ่นยนต์ 4 ขาอย่างง่ายที่ประกอบด้วย 4 เซอร์โวมอเตอร์



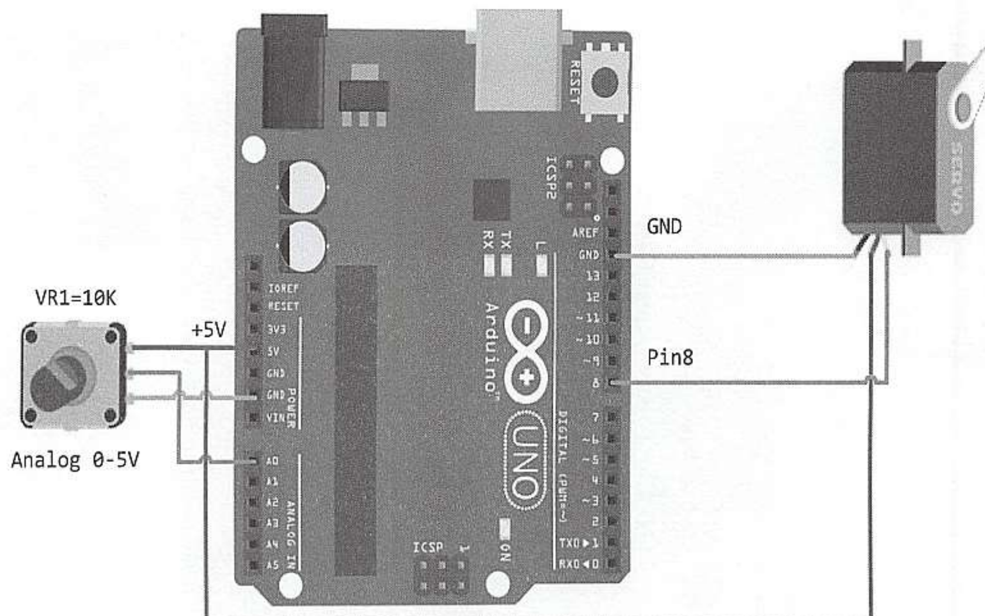
รูปที่ 9.1 หุ่นยนต์ 4 ขาอย่างง่าย

9.1 เซอร์โวมอเตอร์

เซอร์โวมอเตอร์ (Servo Motor) คือมอเตอร์ที่สามารถควบคุมความเร็ว ตำแหน่ง และอัตราเร่งได้ โดยเซอร์โวมอเตอร์จะมีส่วนของวงจรป้อนกลับเพื่อตรวจสอบการทำงาน มีลักษณะควบคุมแบบป้อนกลับ เซอร์โวมอเตอร์มีหลายชนิด แต่ในหัวข้อนี้จะกล่าวถึง RC เซอร์โวมอเตอร์ ซึ่งมีขนาดเล็ก น้ำหนักเบา มีแรงบิดสูง ตัวถังเป็นสี่เหลี่ยม สามารถติดตั้งง่าย หมุนได้ 0–180 องศา หรือครึ่งรอบ สายสัญญาณของ RC เซอร์โวมอเตอร์มี 3 เส้น ประกอบด้วย สัญญาณไฟ กราวด์ และสัญญาณพัลส์ แสดงดังรูปที่ 9.2

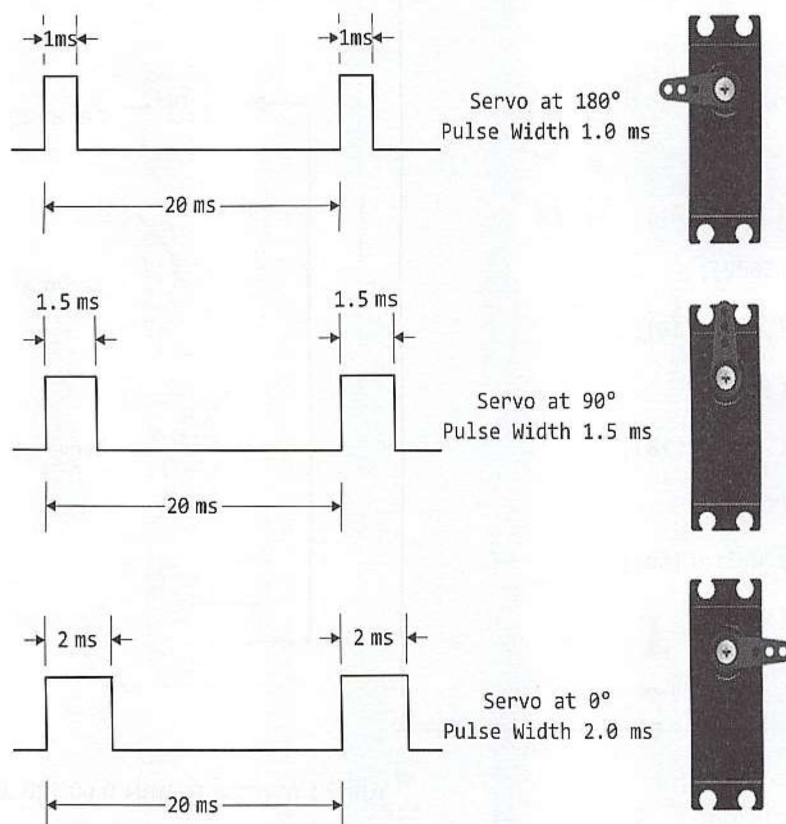


รูปที่ 9.2 RC เซอร์โวมอเตอร์และขาสัญญาณ



รูปที่ 9.3 การเชื่อมต่อ Arduino กับเซอร์โวมอเตอร์

การควบคุมตำแหน่งของเซอร์โวมอเตอร์ ควบคุมได้โดยการปรับความกว้างของสัญญาณพัลส์ ถ้าส่งสัญญาณพัลส์ที่มีความกว้างขนาด 1 ms จะทำให้เซอร์โวมอเตอร์หมุนมาทางซ้ายสุดหรือที่ตำแหน่ง 180 องศา ถ้าส่งสัญญาณพัลส์ 1.5 ms เซอร์โวมอเตอร์จะหมุนมาที่ตำแหน่งตรงกลางหรือที่ตำแหน่ง 90 องศา และถ้าส่งสัญญาณพัลส์ 2 ms เซอร์โวมอเตอร์จะหมุนมาทางขวาสุดหรือที่ตำแหน่ง 0 องศา และต้องส่งสัญญาณพัลส์ทุก ๆ 20 ms เพื่อรักษาตำแหน่งการหมุนของเซอร์โวมอเตอร์ ดังรูปที่ 9.4



รูปที่ 9.4 การควบคุมตำแหน่งของเซอร์โวมอเตอร์

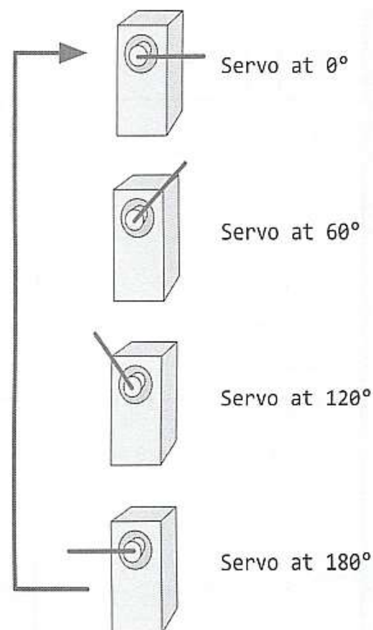
เพื่อความสะดวกในการเขียนโปรแกรมควบคุม สามารถเรียกใช้งานไลบรารี Servo.h ซึ่งเป็นไลบรารีมาตรฐาน และใช้ฟังก์ชันต่าง ๆ เพื่อควบคุมการทำงานของเซอร์โวมอเตอร์ได้

ตัวอย่าง

```
#include <Servo.h>      //เรียกใช้ไลบรารี Servo.h
Servo Servo1;           //ประกาศตัวแปร Servo1
Servo1.attach(8);       //ให้ Servo1 ต่อกับขา 8
Servo1.write(45);       //ให้ Servo1 หมุนไปที่ตำแหน่ง 45 องศา
Servo1.write(75);       //ให้ Servo1 หมุนไปที่ตำแหน่ง 75 องศา
```

ตัวอย่างที่ 9.1 โปรแกรมควบคุมเซอร์โวมอเตอร์ให้หมุน 0 60 120 180 องศา

```
#include <Servo.h>
Servo Servo1;
void setup()
{ Servo1.attach(8);
}
void loop()
{
  Servo1.write(0);
  delay(5000);
  Servo1.write(60);
  delay(1000);
  Servo1.write(120);
  delay(1000);
  Servo1.write(180);
  delay(1000);
}
```



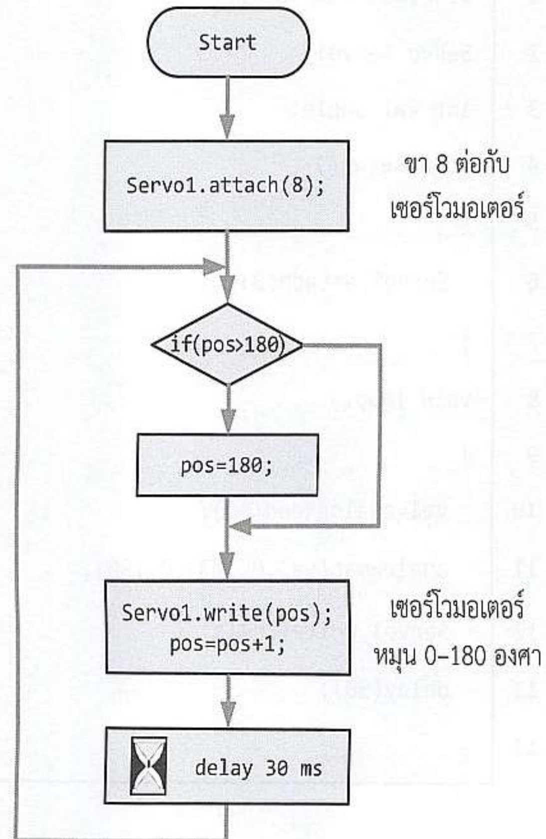
รูปที่ 9.5 การหมุนที่ตำแหน่ง 0 60 120 และ180 องศา

ผลการรันโปรแกรม

โปรแกรมจะวนรอบควบคุมเซอร์โวมอเตอร์ให้หมุน เริ่มจากที่ตำแหน่ง 0 60 120 180 องศา วนรอบหมุนไปเรื่อย ๆ โดยฟังก์ชัน Servo1.write(angle); จะควบคุมให้เซอร์โวมอเตอร์หมุนตามมุมที่ป้อน 0–180 องศา

ตัวอย่างที่ 9.2 โปรแกรมควบคุมเซอร์โวมอเตอร์ให้หมุน 0–180 องศา

```
#include <Servo.h>
Servo Servo1;
int pos;
void setup()
{
  Servo1.attach(8);
}
void loop()
{
  if(pos>180)
  pos=0;
  Servo1.write(pos);
  pos++;
  delay(30);
}
```



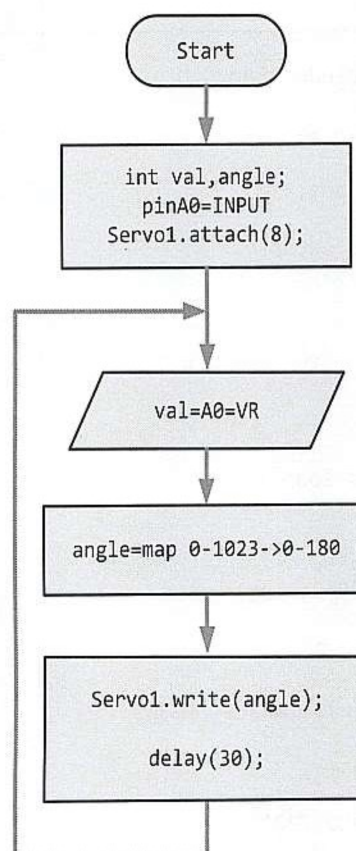
รูปที่ 9.6 โฟลว์ชาร์ตควบคุมเซอร์โวมอเตอร์ให้หมุน 0–180 องศา

ผลการรันโปรแกรม

โปรแกรมจะควบคุมให้เซอร์โวมอเตอร์หมุน เริ่มจากตำแหน่ง 0–180 องศา ตามการเพิ่มของตัวแปร pos ที่มีค่า 0–180 เมื่อเซอร์โวมอเตอร์หมุนไปถึงตำแหน่ง 180 องศาแล้วจะวนรอบเริ่มที่ 0 องศาใหม่

ตัวอย่างที่ 9.3 โปรแกรมควบคุมเซอร์โวมอเตอร์ให้หมุนตามการปรับค่า VR

```
#include <Servo.h>
Servo Servo1;
int val,angle;
void setup()
{
  Servo1.attach(8);
}
void loop()
{
  val=analogRead(A0);
  angle=map(val,0,1023,0,180);
  Servo1.write(angle);
  delay(30);
}
```



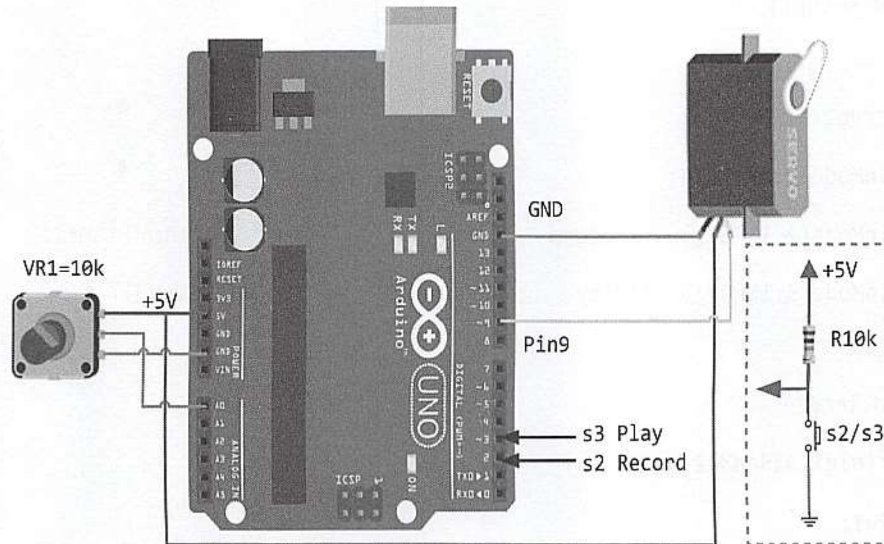
รูปที่ 9.7 โฟลว์ชาร์ตควบคุมเซอร์โวมอเตอร์ตามค่า VR

ผลการรันโปรแกรม

โปรแกรมวนรอบรับค่า VR จากขา A0 มาเก็บไว้ที่ตัวแปร val แล้วทำการแปลงค่าจาก 0–1023 ให้เป็น 0–180 เก็บไว้ในตัวแปร angle เพื่อให้มอเตอร์หมุนไปที่ตำแหน่ง 0–180 องศา ตามฟังก์ชัน Servo1.write(angle);

9.2 การบันทึกตำแหน่งการหมุนของเซอร์โวมอเตอร์

การบันทึกค่าตำแหน่งการหมุนของเซอร์โวมอเตอร์ต้องมีตัวแปรในการเก็บค่า เมื่อกดสวิตช์ s2 (Record) จะบันทึกตำแหน่งหรือมุมในการหมุนของเซอร์โวมอเตอร์ตามการหมุน VR1 แล้วเก็บค่าไว้ในตัวแปรอาร์เรย์ ซึ่งสามารถบันทึกตำแหน่งได้หลายค่าตามการหมุน VR1 และการกดสวิตช์บันทึก s2 สามารถทำซ้ำตำแหน่งเดิมได้เมื่อกดสวิตช์ s3 (Play)



รูปที่ 9.8 วงจรบันทึกตำแหน่งการหมุนของเซอร์โวมอเตอร์

อาร์เรย์ (Array) คือ ตัวแปรแถวลำดับหรือตัวแปรชุดที่เป็นชนิดเดียวกัน ในการประกาศตัวแปรแบบอาร์เรย์จะมีเครื่องหมาย [ ] ต่อท้ายตัวแปรเพื่อบอกขนาดของตัวแปรอาร์เรย์

ตัวอย่าง

int angle[8]; หมายถึง ประกาศตัวแปร angle ให้เป็นตัวแปรอาร์เรย์ มีขนาดเท่ากับ 8 หน่วย ประกอบไปด้วยตัวแปรอาร์เรย์ angle[0] ถึง angle[7] แต่ละตัวจะเก็บข้อมูลได้ 16 บิต



รูปที่ 9.9 การบันทึกตำแหน่งไว้ในตัวแปรอาร์เรย์

ตัวอย่างที่ 9.4 โปรแกรมบันทึกตำแหน่งและการทำงานซ้ำของเซอร์โวมอเตอร์

`#include <Servo.h>`	//เรียกใช้ไลบรารี Servo.h
`Servo Servo1;`	
`int s2,s3,i,x,angle[10];`	//ประกาศตัวแปร
`void setup()`	
`{`	
`  Servo1.attach(9);`	
`  pinMode(9,OUTPUT);`	
`  pinMode(2,INPUT);  //Record`	//ขา 2 เป็นสวิตช์บันทึกตำแหน่ง
`  pinMode(3,INPUT);  //Play`	//ขา 3 เป็นสวิตช์ให้ทำงานซ้ำ
`}`	
`void loop()`	
`{ if(digitalRead(2)==0)`	//ถ้ากดสวิตช์ s2
`  s2=1;`	//ให้ s2 = 1
`  if(digitalRead(2)==1 && s2==1 )`	//ถ้าขา 2 = 1 และ s2 = 1 หรือปล่อยสวิตช์
`  {angle[i]=analogRead(A0);`	//อ่านค่า VR1 จากขา A0
`  angle[i]=map(angle[i],0,1023,0,180);`	//แปลงค่าและบันทึกค่า
`  Serial.println(angle[i]);`	
`  Servo1.write(angle[i]);`	//หมุนมอเตอร์ตามการหมุน VR1
`  delay(1000);`	
`  i++;`	//เลื่อนตำแหน่งอาร์เรย์
`  s2=0;`	//ให้ s2 = 0
`  }`	
`if(digitalRead(3)==0)`	//ถ้ากดสวิตช์ s3
`  s3=1;`	//ให้ s3 = 1
`if(digitalRead(3)==1 && s3==1)`	//ถ้าขา 3 = 1 และ s3 = 1 หรือปล่อยสวิตช์
`{`	
`  for(x=0;x<=i;x++)`	//วนรอบทำซ้ำ
`  { Serial.println(angle[x]);`	
`    Servo1.write(angle[x]);`	//หมุนมอเตอร์ตามค่าที่บันทึก
`    delay(1000);`	

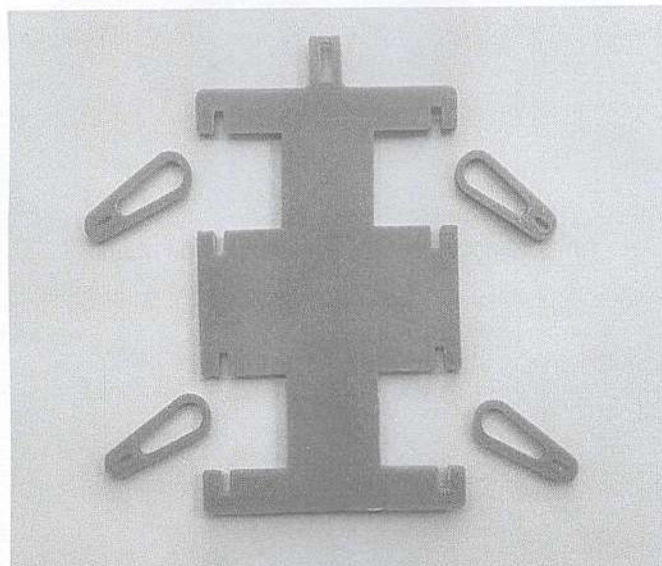
```
      }
      s3=0;              //ให้ s3 = 0 จบการทำซ้ำ
    }
  }
```

ผลการรันโปรแกรม

เมื่อกดสวิตช์ s2 (Record) จะบันทึกตำแหน่งหรือมุมในการหมุนของเซอร์โวมอเตอร์ตามการหมุน VR1 แล้วเก็บค่าไว้ในตัวแปรอาร์เรย์ ค่าตามการหมุน VR1 และการกดสวิตช์บันทึก s2 สามารถทำซ้ำตำแหน่งเดิมได้เมื่อกดสวิตช์ s3 (Play) ซึ่งจะสามารถบันทึกตำแหน่งได้สูงสุด 8 ตำแหน่งตามขนาดของอาร์เรย์

9.3 การสร้างหุ่นยนต์ 4 ขา

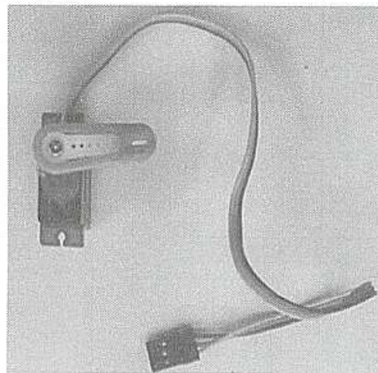
การสร้างหุ่นยนต์ 4 ขาอย่างง่ายจะใช้เซอร์โวมอเตอร์ 4 ตัวในการทำงานแทนขาทั้ง 4 ของหุ่นยนต์ โครงสร้างของตัวหุ่นยนต์ใช้พลาสติกจากการพิมพ์ 3 มิติ หรืออาจจะใช้กระดาษแข็งแทนก็ได้ (สามารถติดต่อขอไฟล์แบบร่างได้ที่ Facebook : Adon_plearnplus)



รูปที่ 9.10 โครงสร้างลำตัวและขาของหุ่นยนต์ 4 ขา

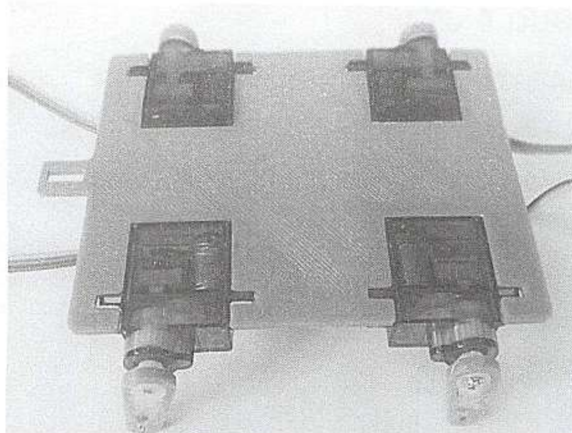
หุ่นยนต์ 4 ขาอย่างง่ายมีขั้นตอนการสร้างและประกอบดังนี้

1) ประกอบส่วนขาเข้ากับเซอร์โวมอเตอร์ทั้ง 4 ตัว ดังรูปที่ 9.11



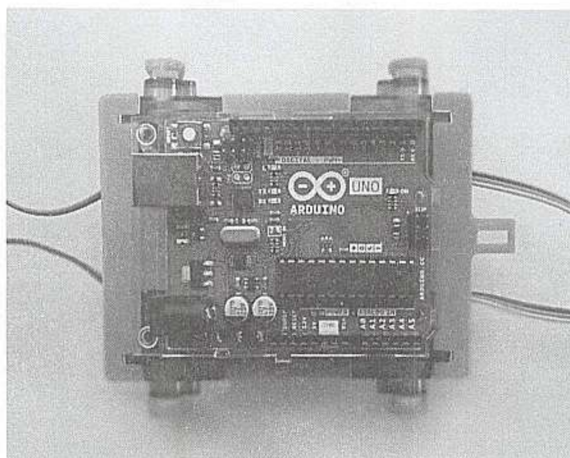
รูปที่ 9.11 การประกอบส่วนขาเข้ากับเซอร์โวมอเตอร์

2) ประกอบขาหุ่นยนต์และเซอร์โวมอเตอร์เข้ากับตัวหุ่นยนต์ ดังรูปที่ 9.12



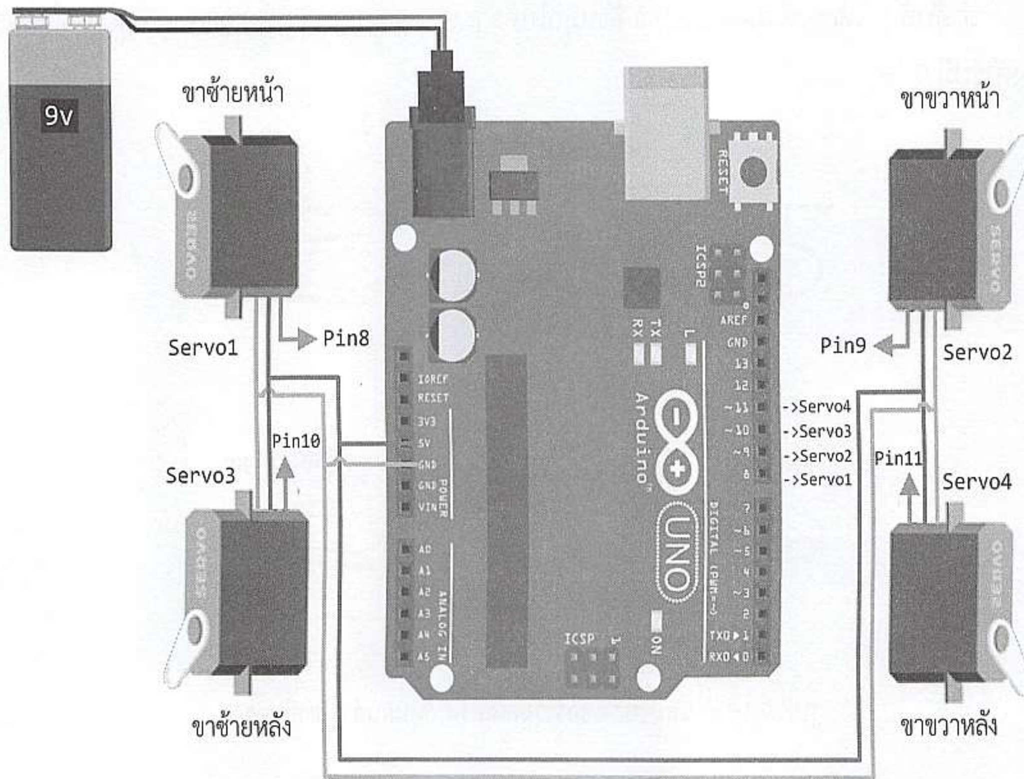
รูปที่ 9.12 การประกอบเซอร์โวมอเตอร์เข้ากับตัวหุ่นยนต์

3) ติดตั้งบอร์ด Arduino เข้ากับตัวหุ่นยนต์ ดังรูปที่ 9.13



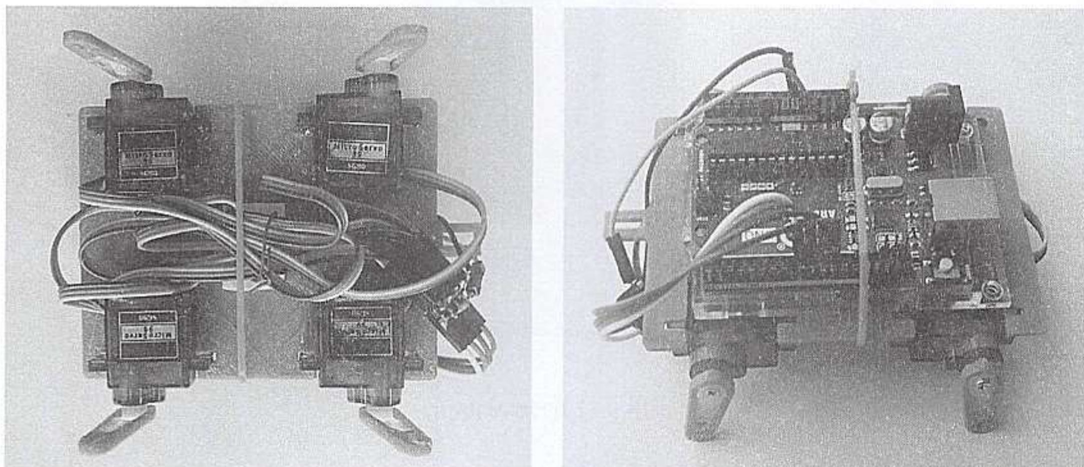
รูปที่ 9.13 การติดตั้งบอร์ด Arduino เข้ากับตัวหุ่นยนต์

4) ต่อสายสัญญาณขา 8 9 10 และ 11 เข้ากับเซอร์โวมอเตอร์ตัวที่ 1 2 3 และ 4 ต่อสัญญาณไฟและกราวด์ของมอเตอร์ทั้ง 4 ร่วมกัน ดังรูปที่ 9.14 (ในการทดลองอาจใช้แหล่งจ่ายไฟจากพอร์ต USB)



รูปที่ 9.14 วงจรหุ่นยนต์ 4 ขาอย่างง่าย

5) เก็บสายเซอร์โวมอเตอร์และสายสัญญาณไว้ข้างใต้ของตัวหุ่นยนต์ให้เรียบร้อย ดังรูปที่ 9.15

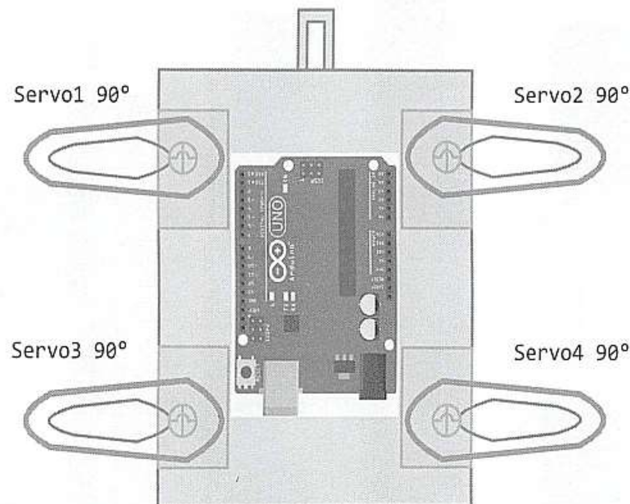


รูปที่ 9.15 การเก็บสายสัญญาณและหุ่นยนต์ที่ประกอบเสร็จ

9.4 การเดินของหุ่นยนต์ 4 ขา

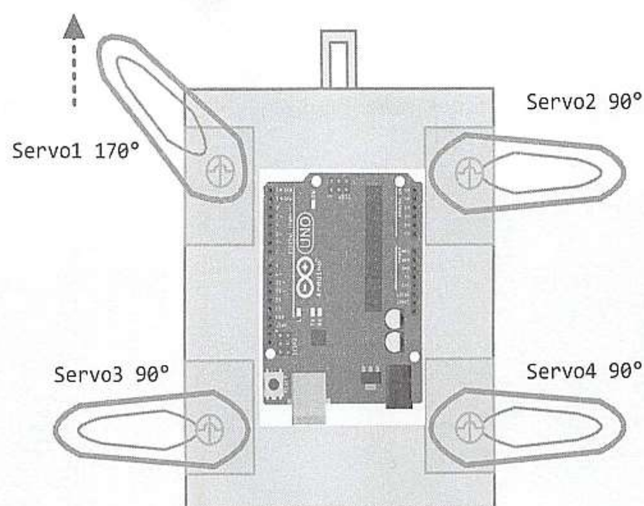
การเดินหน้าของหุ่นยนต์ 4 ขาทำได้โดยควบคุมการทำงานของเซอร์โวมอเตอร์ โดยเริ่มจากให้หุ่นยนต์ยืนตรงแล้วก้าวขาซ้ายหน้า ขวาหน้า ซ้ายหลัง และขวาหลัง ตามจังหวะการเดิน 5 ลำดับดังนี้

ลำดับที่ 1) ให้เซอร์โวมอเตอร์ทั้ง 4 ตัวหมุนไปที่ตำแหน่ง 90 องศา ซึ่งเป็นตำแหน่งที่หุ่นยนต์ยืนตรง แสดงดังรูปที่ 9.16



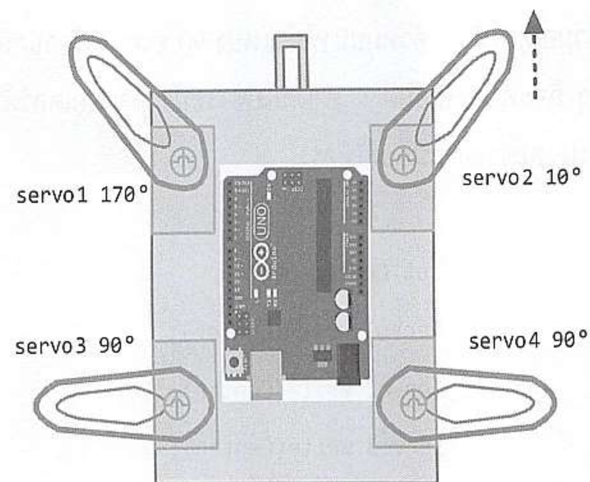
รูปที่ 9.16 ตำแหน่งของเซอร์โวมอเตอร์เมื่อหุ่นยนต์ 4 ขายืนตรง

ลำดับที่ 2) ให้ Servo1 ขาซ้ายหน้าหมุนไปด้านหน้าที่ตำแหน่ง 90 + 80 = 170 องศา แสดงดังรูปที่ 9.17



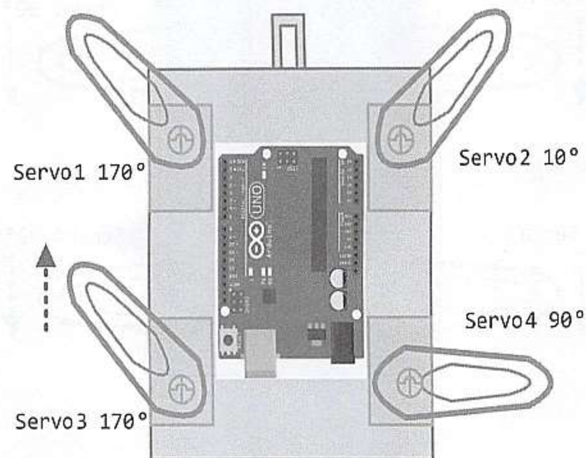
รูปที่ 9.17 ก้าวขาซ้ายหน้า

ลำดับที่ 3) ให้ Servo2 ขาขวาหน้าหมุนไปด้านหน้าที่ตำแหน่ง 90 – 80 = 10 องศา แสดงดังรูปที่ 9.18 การหมุนไปด้านหน้าและด้านหลัง ตำแหน่งหรือมุมจะไม่เท่ากัน (ตรงข้ามกัน)



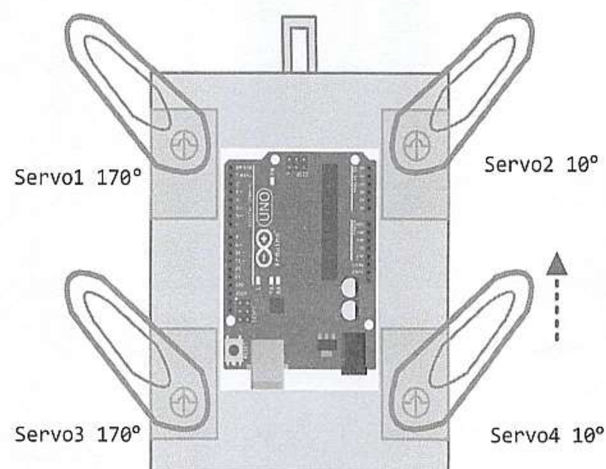
รูปที่ 9.18 ก้าวขาขวาหน้า

ลำดับที่ 4) ให้ Servo3 ขาซ้ายหลังหมุนไปด้านหน้าที่ตำแหน่ง 170 องศา แสดงดังรูปที่ 9.19



รูปที่ 9.19 ก้าวขาซ้ายหลังไปด้านหน้า

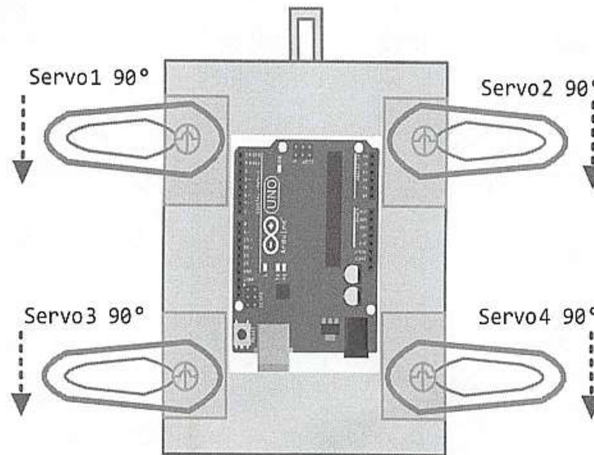
ลำดับที่ 5) ให้ Servo4 ขาขวาหลังหมุนไปด้านหน้าที่ตำแหน่ง 10 องศา แสดงดังรูปที่ 9.20



รูปที่ 9.20 ก้าวขาขวาหลังไปด้านหน้า

จากนั้นให้เซอร์โวมอเตอร์ทั้ง 4 ตัวหมุนมาที่ตำแหน่ง 90 องศา หรือวนกลับมายืนตรงใหม่ แล้ววนรอบการทำงานไปเรื่อย ๆ ก็จะทำให้หุ่นยนต์ 4 ขาเดินไปข้างหน้า หากมอเตอร์ตัวใดไม่หมุนมาที่ตำแหน่งตรงกลาง ต้องปรับตำแหน่งและขาของหุ่นยนต์ให้ตรง

```
int Center=90;
Servo1.write(Center);
Servo2.write(Center);
Servo3.write(Center);
Servo4.write(Center);
```



รูปที่ 9.21 หุ่นยนต์ 4 ขากลับมายืนตรง

ตัวอย่างที่ 9.5 โปรแกรมรีเซ็ตหุ่นยนต์ 4 ขาให้ยืนตรง

```
#include <Servo.h>
Servo Servo1; //Front Left
Servo Servo2; //Front Right
Servo Servo3; //Rear Left
Servo Servo4; //Rear Right
int Center = 90;              //กำหนดมุมการหมุนที่ 90 องศา ให้หุ่นยนต์เดินตรง
void setup()
{
  Servo1.attach(8);           //ให้ Servo1 ต่อกับขา 8
  Servo2.attach(9);           //ให้ Servo2 ต่อกับขา 9
  Servo3.attach(10);          //ให้ Servo3 ต่อกับขา 10
  Servo4.attach(11);          //ให้ Servo4 ต่อกับขา 11
}
void loop()
{
  Servo1.write(Center);       //ให้ Servo1 หมุนมาที่ตำแหน่ง 90 องศา
  Servo2.write(Center);       //ให้ Servo2 หมุนมาที่ตำแหน่ง 90 องศา
  Servo3.write(Center);       //ให้ Servo3 หมุนมาที่ตำแหน่ง 90 องศา
  Servo4.write(Center);       //ให้ Servo4 หมุนมาที่ตำแหน่ง 90 องศา
}
```

ผลการรันโปรแกรม

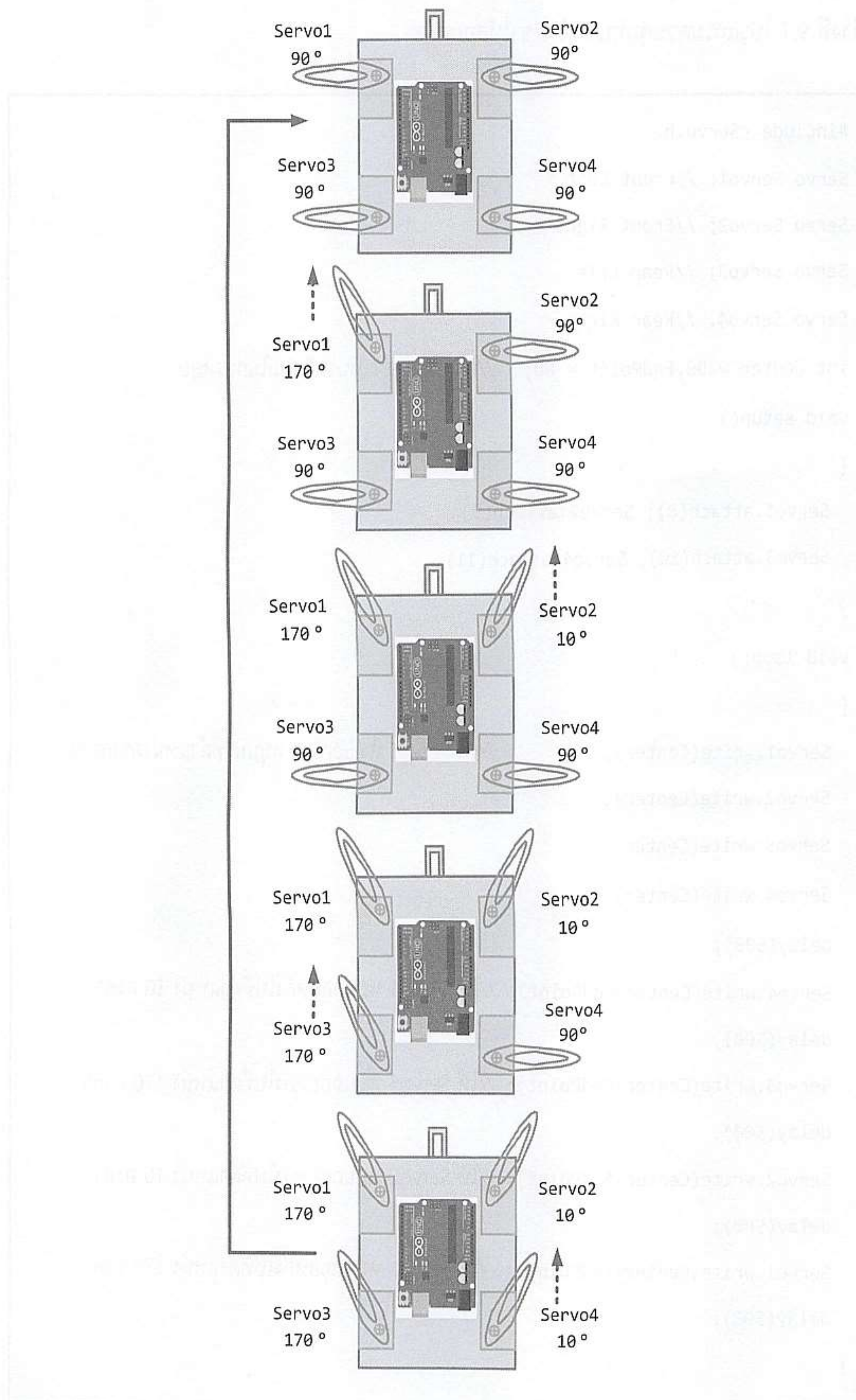
โปรแกรมจะควบคุมให้เซอร์โวมอเตอร์ทั้ง 4 ตัวหมุนไปที่ตำแหน่ง 90 องศา ทำให้หุ่นยนต์ 4 ขายืนตรง โดย Servo1 ถึง Servo4 จะต่อเข้ากับขา 8–11 ตามลำดับ หากมอเตอร์ตัวใดไม่หมุนมาที่ตำแหน่งตรงกลาง (90 องศา) ต้องปรับตำแหน่งและขาของหุ่นยนต์ให้ตรง

ตัวอย่างที่ 9.6 โปรแกรมควบคุมหุ่นยนต์ 4 ขาให้เดินหน้า

```
#include <Servo.h>
Servo Servo1; //Front Left
Servo Servo2; //Front Right
Servo Servo3; //Rear Left
Servo Servo4; //Rear Right
int Center = 90,EndPoint = 80;              //กำหนดค่าตำแหน่งการเดิน
void setup()
{ Servo1.attach(8); Servo2.attach(9);       //ให้ Servo1 Servo2 ต่อกับขา 8 และ 9
  Servo3.attach(10); Servo4.attach(11);     //ให้ Servo3 Servo4 ต่อกับขา 10 และ 11
}
void loop()
{ Servo1.write(Center);          //ให้ Servo1 ถึง Servo4 หมุนมาที่ตำแหน่งตรงกลาง
  Servo2.write(Center);
  Servo3.write(Center);
  Servo4.write(Center);
  delay(500);
  Servo1.write(Center-EndPoint); //ให้ Servo1 หมุนไปหน้า (ตำแหน่ง 90 – 80 = 10 องศา)
  delay(500);
  Servo2.write(Center+EndPoint); //ให้ Servo2 หมุนไปหน้า (ตำแหน่ง 90 + 80 = 170 องศา)
  delay(500);
  Servo3.write(Center-EndPoint); //ให้ Servo3 หมุนไปหน้า (ตำแหน่ง 90 – 80 = 10 องศา)
  delay(500);
  Servo4.write(Center+EndPoint); //ให้ Servo4 หมุนไปหน้า (ตำแหน่ง 90 + 80 = 170 องศา)
  delay(2000);
}
```

ผลการรันโปรแกรม

โปรแกรมจะควบคุมให้เซอร์โวมอเตอร์ทั้ง 4 หมุนเป็นจังหวะตามลำดับที่ 1 ถึง 5 ผลคือทำให้หุ่นยนต์ 4 ขาเดินหน้า ขั้นตอนการเดินของหุ่นยนต์แสดงดังรูปที่ 9.22



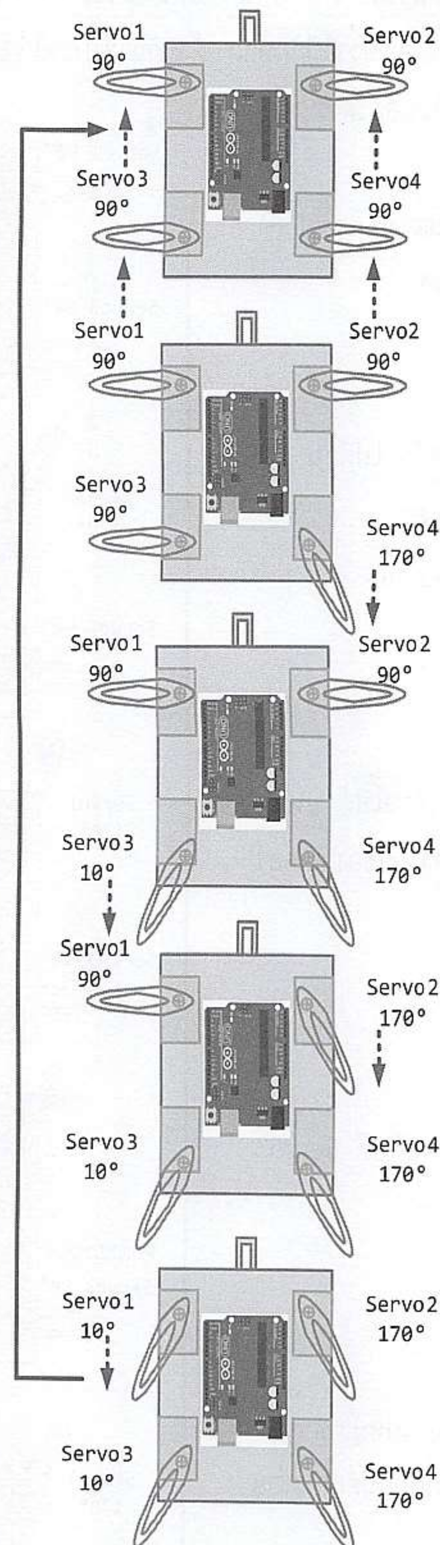
รูปที่ 9.22 ขั้นตอนการเดินหน้าของหุ่นยนต์ 4 ขา

ตัวอย่างที่ 9.7 โปรแกรมควบคุมหุ่นยนต์ 4 ขาให้ถอยหลัง

```
#include <Servo.h>
Servo Servo1; //Front Left
Servo Servo2; //Front Right
Servo Servo3; //Rear Left
Servo Servo4; //Rear Right
int Center = 90,EndPoint = 80;   //กำหนดค่าตำแหน่งหรือมุมในการหมุน
void setup()
{
  Servo1.attach(8); Servo2.attach(9);
  Servo3.attach(10); Servo4.attach(11);
}
void loop()
{
  Servo1.write(Center);             //ให้ Servo1 ถึง Servo4 หมุนมาที่ตำแหน่งตรงกลาง
  Servo2.write(Center);
  Servo3.write(Center);
  Servo4.write(Center);
  delay(500);
  Servo4.write(Center-EndPoint); //ให้ Servo4 หมุนถอยหลังไปที่ตำแหน่ง 10 องศา
  delay(500);
  Servo3.write(Center+EndPoint); //ให้ Servo3 หมุนถอยหลังไปที่ตำแหน่ง 170 องศา
  delay(500);
  Servo2.write(Center-EndPoint); //ให้ Servo2 หมุนถอยหลังไปที่ตำแหน่ง 10 องศา
  delay(500);
  Servo1.write(Center+EndPoint); //ให้ Servo1 หมุนถอยหลังไปที่ตำแหน่ง 170 องศา
  delay(500);
}
```

ผลการรันโปรแกรม

โปรแกรมจะควบคุมให้เซอร์โวมอเตอร์ทั้ง 4 หมุนตามลำดับที่ 1 ถึงลำดับที่ 5 ทำให้หุ่นยนต์ 4 ขาเดินถอยหลัง แสดงขั้นตอนการเดินของหุ่นยนต์ดังรูปที่ 9.23



รูปที่ 9.23 ขั้นตอนการเดินถอยหลังของหุ่นยนต์ 4 ขา

9.5 การเลี้ยวซ้ายหรือเลี้ยวขวาของหุ่นยนต์ 4 ขา

การเลี้ยวซ้ายของหุ่นยนต์มีขั้นตอนคล้ายกับการเดินหน้าปกติ เพียงแต่ให้ Servo3 หยุดอยู่กับที่จะทำให้หุ่นยนต์เลี้ยวซ้าย ส่วนการเลี้ยวขวาก็มีลักษณะคล้ายกัน เพียงแต่ให้ Servo4 หยุดอยู่กับที่ การเลี้ยวซ้ายมีขั้นตอนการเคลื่อนที่ 5 ลำดับ แสดงดังรูปที่ 9.24

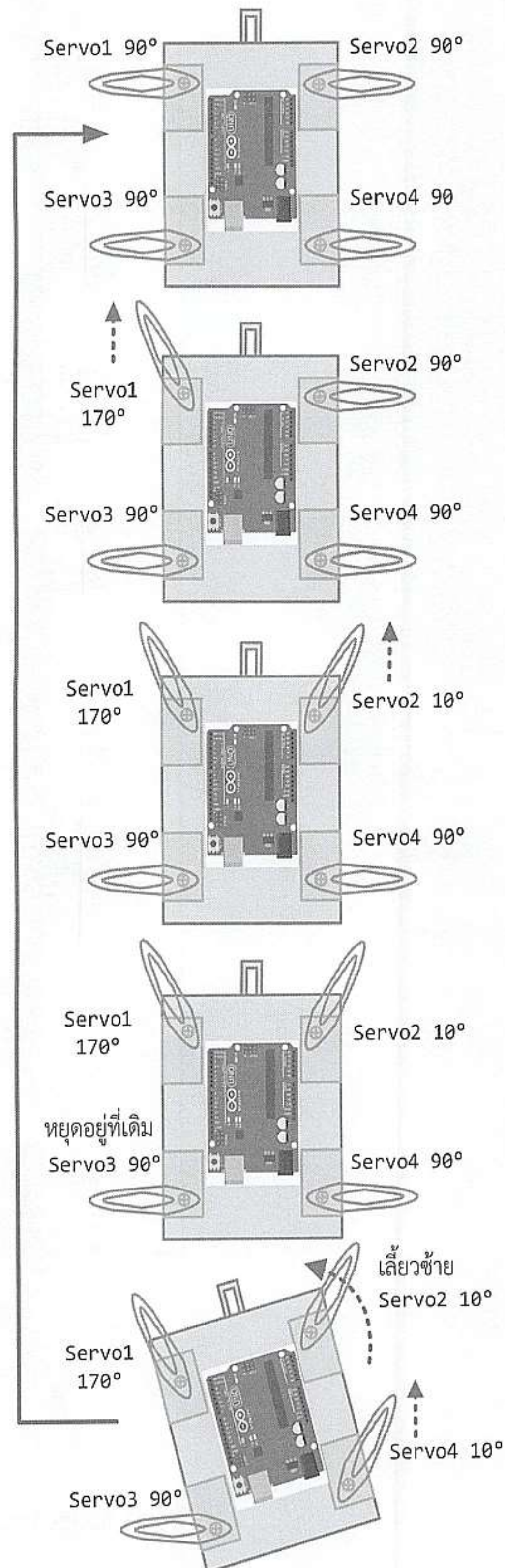
ลำดับที่ 1) ให้หุ่นยนต์ยืนตรง Servo1 ถึง Servo4 หมุนมาที่ตำแหน่ง 90 องศา

ลำดับที่ 2) ขาซ้ายหน้าก้าวไปด้านหน้า Servo1 หมุนมาที่ตำแหน่ง 170 องศา

ลำดับที่ 3) ขาขวาหน้าก้าวไปด้านหน้า Servo2 หมุนมาที่ตำแหน่ง 10 องศา (มุมของมอเตอร์ซ้ายและขวาจะตรงข้ามกัน)

ลำดับที่ 4) ขาซ้ายหลังอยู่กับที่ Servo3 อยู่ที่ตำแหน่ง 90 องศา

ลำดับที่ 5) ขาขวาหลังหมุนไปหน้า Servo4 หมุนไปที่ตำแหน่ง 10 องศา ทำให้หุ่นยนต์เลี้ยวซ้าย



รูปที่ 9.24 ขั้นตอนการเลี้ยวซ้ายของหุ่นยนต์ 4 ขา

ตัวอย่างที่ 9.8 โปรแกรมควบคุมหุ่นยนต์ 4 ขาให้เลี้ยวซ้าย

```
#include <Servo.h>
Servo Servo1; //Front Left
Servo Servo2; //Front Right
Servo Servo3; //Rear Left
Servo Servo4; //Rear Right
int Center = 90,EndPoint = 80;   //กำหนดค่าตำแหน่งการเดิน
void setup()
{ Servo1.attach(8); Servo2.attach(9);    //ให้ Servo1 Servo 2 ต่อกับขา 8 และ 9
  Servo3.attach(10); Servo4.attach(11);  //ให้ Servo3 Servo 4 ต่อกับขา 10 และ 11
}
void loop()
{ Servo1.write(Center);            //ให้ Servo1 ถึง Servo4 หมุนมาที่ตำแหน่งตรงกลาง
  Servo2.write(Center);
  Servo3.write(Center);
  Servo4.write(Center);
  delay(500);
  Servo1.write(Center+EndPoint);  //ให้ Servo1 หมุนไปหน้า (ตำแหน่ง 90 + 80 = 170 องศา)
  delay(500);
  Servo2.write(Center-EndPoint); //ให้ Servo2 หมุนไปหน้า (ตำแหน่ง 90 – 80 = 10 องศา)
  delay(500);
  Servo3.write(Center-0);          //ให้ Servo3 อยู่ตำแหน่งเดิมที่ 90 องศา
  delay(500);
  Servo4.write(Center-EndPoint); //ให้ Servo4 หมุนไปหน้า (ตำแหน่ง 90 – 80 = 10 องศา)
  delay(2000);
}
```

ผลการรันโปรแกรม

การเลี้ยวซ้ายของหุ่นยนต์มีขั้นตอนคล้ายกับการเดินหน้าปกติ เพียงแต่ให้ Servo3 หยุดอยู่กับที่ก็จะทำให้หุ่นยนต์เลี้ยวซ้าย ส่วนการเลี้ยวขวาก็มีลักษณะคล้ายกัน เพียงแต่ให้ Servo4 หยุดอยู่กับที่ โปรแกรมจะควบคุมให้เซอร์โวมอเตอร์ทั้ง 4 หมุนไปตามลำดับที่ 1 ถึงลำดับที่ 5 ทำให้หุ่นยนต์ 4 ขาเลี้ยวซ้าย ขั้นตอนการเดินของหุ่นยนต์แสดงดังรูปที่ 9.24

ตัวอย่างที่ 9.9 โปรแกรมควบคุมหุ่นยนต์ 4 ขาให้เลี้ยวขวา

```
#include <Servo.h>
Servo Servo1; //Front Left
Servo Servo2; //Front Right
Servo Servo3; //Rear Left
Servo Servo4; //Rear Right
int Center = 90;                  //กำหนดค่าตำแหน่งตรงกลาง 90 องศา
int EndPoint = 80;                //กำหนดค่าตำแหน่งการก้าว
void setup()
{
  Servo1.attach(8);               //ให้ Servo1 ต่อกับขา 8
  Servo2.attach(9);               //ให้ Servo2 ต่อกับขา 9
  Servo3.attach(10);              //ให้ Servo3 ต่อกับขา 10
  Servo4.attach(11);              //ให้ Servo4 ต่อกับขา 11
}
void loop()
{
  Servo1.write(Center);           //ให้ Servo1 หมุนมาที่ตำแหน่ง 90 องศา
  Servo2.write(Center);           //ให้ Servo2 หมุนมาที่ตำแหน่ง 90 องศา
  Servo3.write(Center);           //ให้ Servo3 หมุนมาที่ตำแหน่ง 90 องศา
  Servo4.write(Center);           //ให้ Servo4 หมุนมาที่ตำแหน่ง 90 องศา หุ่นยนต์ยืนตรง
  delay(500);
  Servo1.write(Center+EndPoint);  //ให้ Servo1 หมุนไปหน้า (ตำแหน่ง 90 + 80 = 170 องศา)
  delay(500);
  Servo2.write(Center-EndPoint);  //ให้ Servo2 หมุนไปหน้า (ตำแหน่ง 90 – 80 = 10 องศา)
  delay(500);
  Servo4.write(Center+0);         //ให้ Servo4 อยู่ตำแหน่งเดิมที่ 90 องศา
  delay(500);
  Servo3.write(Center+EndPoint);  //ให้ Servo3 หมุนไปหน้า (ตำแหน่ง 90 + 80 = 170 องศา)
  delay(2000);                    //หน่วงเวลา 2000 ms
}
```

ผลการรันโปรแกรม

โปรแกรมจะควบคุมให้หุ่นยนต์เลี้ยวขวา การเลี้ยวขวาของหุ่นยนต์มีขั้นตอนคล้ายกับการเลี้ยวซ้าย แต่ต่างกันที่ให้ Servo4 หยุดอยู่กับที่ก็จะทำให้หุ่นยนต์เลี้ยวขวา ซึ่งมี 5 ลำดับขั้นตอน แสดงดังรูปที่ 9.25

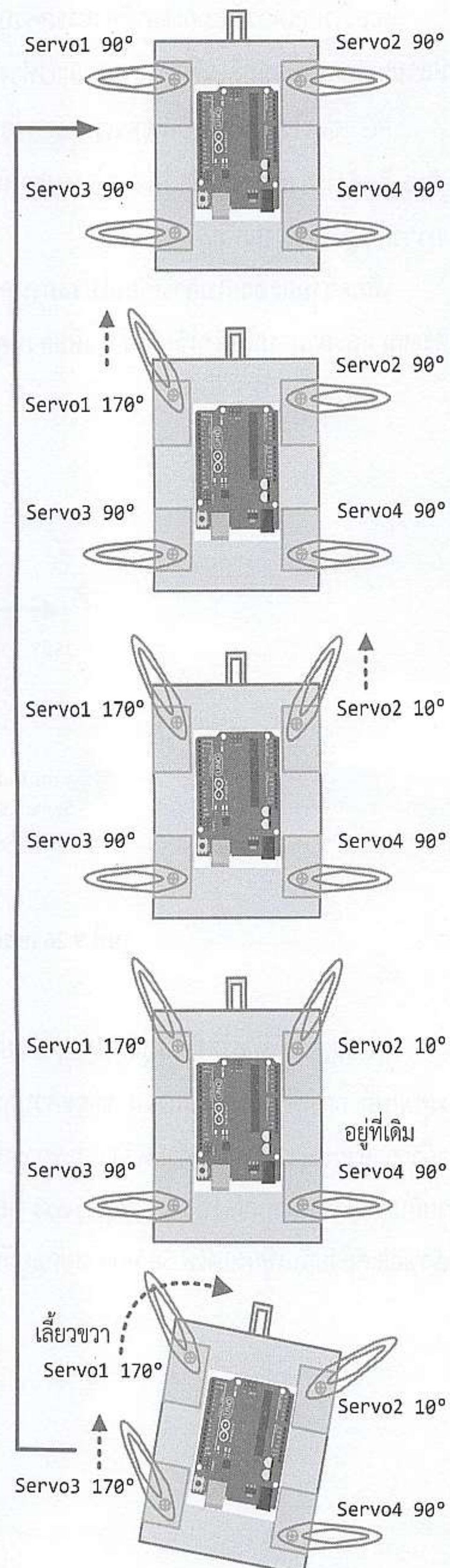
ลำดับที่ 1) ให้หุ่นยนต์ยืนตรง Servo1 ถึง Servo4 หมุนมาที่ตำแหน่ง 90 องศา

ลำดับที่ 2) ขาซ้ายหน้าก้าวไปด้านหน้า Servo1 หมุนมาที่ตำแหน่ง 170 องศา

ลำดับที่ 3) ขาขวาหน้าก้าวไปด้านหน้า Servo2 หมุนมาที่ตำแหน่ง 10 องศา (มุมของมอเตอร์ซ้ายและขวาจะตรงข้ามกัน)

ลำดับที่ 4) ขาขวาหลังอยู่กับที่ Servo4 อยู่ที่ตำแหน่ง 90 องศา

ลำดับที่ 5) ขาซ้ายหลังหมุนไปหน้า Servo3 หมุนไปที่ตำแหน่ง 170 องศา ทำให้หุ่นยนต์เลี้ยวขวา



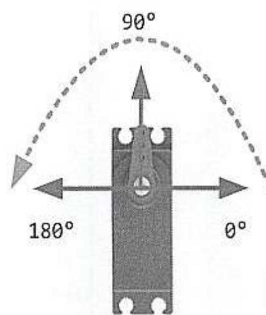
รูปที่ 9.25 ขั้นตอนการเลี้ยวขวาของหุ่นยนต์ 4 ขา

9.6 สรุป

เซอร์โวมอเตอร์ คือมอเตอร์ที่สามารถควบคุมความเร็ว ตำแหน่ง และอัตราเร่งได้ โดยเซอร์โวมอเตอร์จะมีส่วนของวงจรป้อนกลับเพื่อตรวจสอบการทำงาน

RC เซอร์โวมอเตอร์เป็นเซอร์โวมอเตอร์ชนิดหนึ่ง มีขนาดเล็ก น้ำหนักเบา มีแรงบิดสูง ตัวถังเป็นสี่เหลี่ยม ติดตั้งง่าย สามารถหมุนได้ 0–180 องศาหรือครึ่งรอบ มีสายสัญญาณ 3 เส้นประกอบไปด้วยสัญญาณไฟ กราวด์ และสัญญาณพัลส์

เพื่อความสะดวกในการเขียนโปรแกรมควบคุม ให้เรียกใช้งานไลบรารี Servo.h ซึ่งเป็นไลบรารีมาตรฐาน และสามารถใช้ฟังก์ชันต่าง ๆ เพื่อควบคุมการทำงานของเซอร์โวมอเตอร์ได้



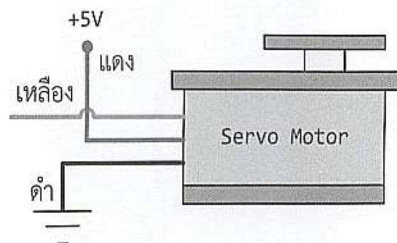

```
#include <Servo.h>
Servo servo1;
servo1.write(0-180)
```

รูปที่ 9.26 การควบคุมเซอร์โวมอเตอร์

หุ่นยนต์ 4 ขาที่สร้างเป็นหุ่นยนต์ 4 ขาอย่างง่าย ใช้เซอร์โวมอเตอร์ 4 ตัวในการทำงานแทนขาทั้ง 4 ของหุ่นยนต์ การเดินของหุ่นยนต์ 4 ขาจะควบคุมเซอร์โวมอเตอร์ให้ทำงานโดยเริ่มจากให้หุ่นยนต์ยืนตรงแล้วก้าวขาซ้ายหน้า ขวาหน้า ซ้ายหลัง และขวาหลังตามจังหวะการเดิน การเลี้ยวซ้ายของหุ่นยนต์มีขั้นตอนคล้ายกับการเดินหน้าปกติ เพียงแต่ให้ Servo3 หยุดอยู่กับที่ ก็จะทำให้หุ่นยนต์เลี้ยวซ้าย ส่วนการเลี้ยวขวาก็มีลักษณะคล้ายกันเพียงแต่ให้ Servo4 หยุดอยู่กับที่

แบบฝึกหัดบทที่ 9

1. อธิบายขาสัญญาณของเซอร์โวมอเตอร์

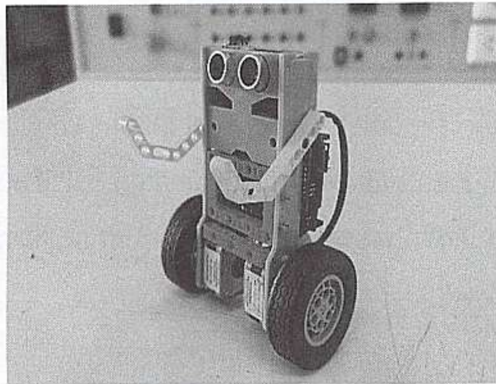


รูปที่ 9.27 ขาสัญญาณของเซอร์โวมอเตอร์

2. เขียนโปรแกรมควบคุมเซอร์โวมอเตอร์ให้หมุน 0–180 องศา และหมุนครั้งละ 10 องศา
3. เขียนโปรแกรมควบคุมเซอร์โวมอเตอร์ให้จำตำแหน่งได้ 12 ตำแหน่งตามการบันทึก
4. อธิบายการเดินหน้าของหุ่นยนต์
5. อธิบายการถอยหลังของหุ่นยนต์
6. เขียนโปรแกรมควบคุมหุ่นยนต์ให้เดินหน้า 1 นาที และถอยหลัง 1 นาที
7. เขียนโปรแกรมควบคุมหุ่นยนต์ให้เลี้ยวซ้าย 1 นาที และเลี้ยวขวา 1 นาที

บทที่

10 หุ่นยนต์ 2 ล้อสมดุล

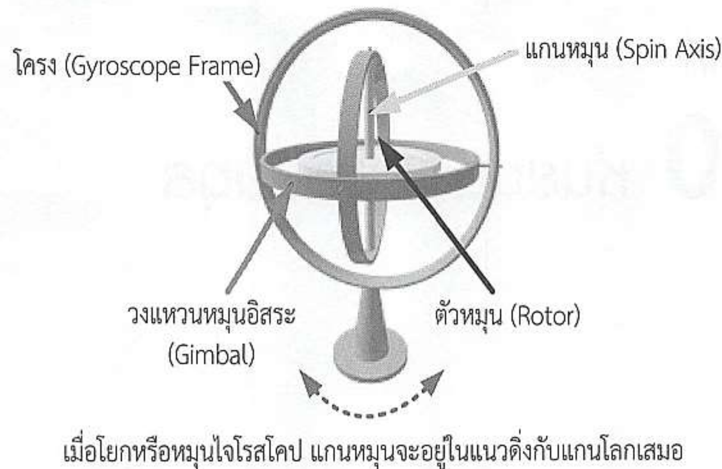


หุ่นยนต์ 2 ล้อสมดุล (Balancing Robot) คือหุ่นยนต์ที่ใช้ล้อในการเคลื่อนที่เพียง 2 ล้อ ฐานของหุ่นยนต์ใช้พื้นที่น้อยกว่าหุ่นยนต์ที่มีมากกว่า 2 ล้อ ทำให้สามารถเคลื่อนที่ได้สะดวก หลักการทำงานของหุ่นยนต์ 2 ล้อ จะมีไมโครคอนโทรลเลอร์ Arduino ทำหน้าที่ควบคุมสเต็ปเปอร์มอเตอร์ให้เดินหน้าหรือถอยหลังตามการเอียงของหุ่นยนต์ โดยใช้ระบบควบคุมแบบ PID คำนวณหาค่าทิศทางและความเร็วในการหมุนของสเต็ปเปอร์มอเตอร์เพื่อให้หุ่นยนต์ 2 ล้อสามารถทรงตัวอยู่ได้

10.1 ไจโรสโคปและอุปกรณ์วัดความเร่ง

ไจโรสโคป (Gyroscope) คืออุปกรณ์ที่ใช้วัดระดับและทิศทางของแกนหมุนตามแนวดิ่งของแกนโลก โดยอาศัยหลักการของแรงเฉื่อยเพื่อรักษาทิศทางของแกนหมุนให้อยู่ในตำแหน่งเดิมตลอดเวลา โครงสร้างของไจโรสโคปแสดงดังรูปที่ 10.1

หลักการทำงานคือ แกนของไจโรสโคป (Spin Axis) จะอยู่ในแนวดิ่งตามแกนโลก ไม่ว่าไจโรสโคปจะหมุนไปในทิศทางใด ซึ่งอาศัยหลักการนี้ในการวัดตำแหน่งการเคลื่อนที่ของวัตถุ เช่น โดรน เครื่องบิน รวมถึงในสมาร์ทโฟน

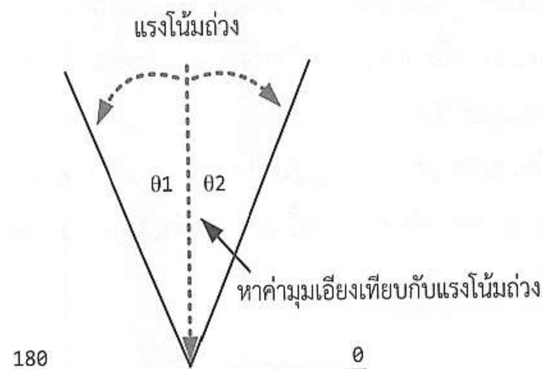


รูปที่ 10.1 ไจโรสโคป

อุปกรณ์วัดความเร่ง (Accelerometer) ในที่นี้หมายถึงเซนเซอร์ที่ใช้วัดความเร่งหรือตรวจจับอัตราการเปลี่ยนแปลงความเร็วแบบ 3 แกน (3-Axis Accelerometer) ซึ่งใช้สำหรับตรวจจับการเคลื่อนที่หรือการเอียงของวัตถุไปจากแนวแกน การประยุกต์ที่คุ้นเคยกันดีคืออุปกรณ์ในสมาร์ทโฟนที่ใช้ตรวจจับการเอียงหรือการเขย่าเครื่องนั่นเอง

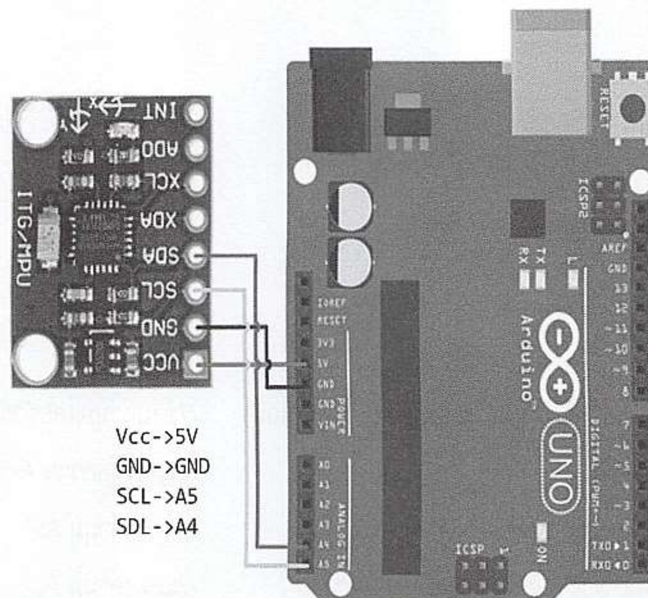
10.2 โมดูล GY-521 MPU6050

โมดูล GY-521 MPU6050 เป็นโมดูลหรือบอร์ดอิเล็กทรอนิกส์ที่ประกอบด้วย Accelerometer และไจโรสโคปอยู่ในตัวเดียวกัน ชิปที่ใช้ประมวลผลคือ MPU6050 สามารถใช้วัดได้ทั้งมุมและความเร่งใน 3 แกน คือ x, y และ z มีการเชื่อมต่อแบบ I2C โดยใช้สายสัญญาณเพียง 2 เส้น หลักการวัดมุมคือ เมื่อมีการหมุนเอียงหรือเคลื่อนที่ตำแหน่งของเซนเซอร์ จะเกิดความเร่งของวัตถุที่เคลื่อนที่เทียบกับแรงโน้มถ่วงของโลก (Gravity) แล้วจึงนำมาคำนวณเป็นค่ามุม



รูปที่ 10.2 การหาค่ามุมเอียงเทียบกับแรงโน้มถ่วงของโลก

การใช้งานสามารถเขียนโปรแกรมเพื่อเข้าถึงเรจิสเตอร์ของเซนเซอร์ หรือเรียกใช้ไลบรารี MPU6050.h เพื่อให้อ่านค่ามุมได้ง่าย โดยที่ Ax Ay และ Az เป็นค่ามุมตามการเอียงของโมดูล ส่วน Gx Gy และ Gz คือค่าไจโร (Gyro) เป็นค่าที่เกิดจากความเร่งในการเคลื่อนที่ของวัตถุซึ่งจะมีค่าเปลี่ยนไปเมื่อมีการเคลื่อนที่ และจะมีค่าใกล้เคียง 90 องศาเมื่อวัตถุหยุดนิ่ง



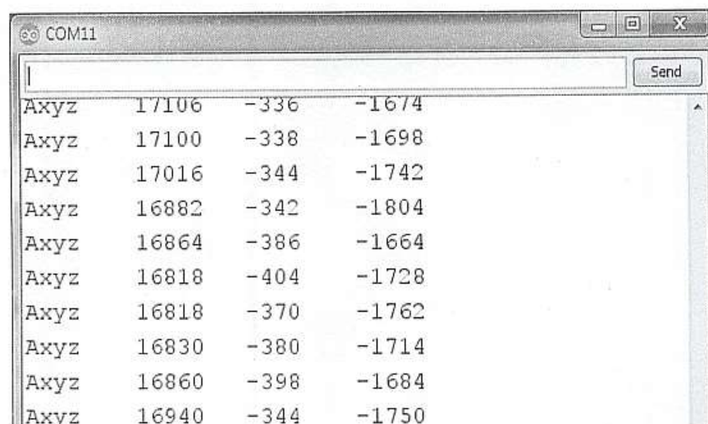
รูปที่ 10.3 การเชื่อมต่อ Arduino กับ GY-521 MPU6050

ตัวอย่างที่ 10.1 โปรแกรมอ่านค่ามุม -18000 ถึง +18000 ในแนวแกน x y และ z

```
#include <MPU6050.h>                                    //เรียกใช้ไลบรารี MPU6050.h
MPU6050 mpu;
int Ax, Ay, Az;                                         //ประกาศตัวแปร Ax Ay Az
void setup()
{
  Serial.begin(9600);                                   //กำหนดอัตราการสื่อสารแบบอนุกรม
  mpu.initialize();                                     //เริ่มการทำงาน MPU6050
}
void loop()
{
  mpu.getMotion6(&Ax, &Ay, &Az, &Gx, &Gy, &Gz);         //อ่านค่ามุมและความเร่ง
  Serial.print("Axyz \t");                              //แสดงข้อความ Axyz
  Serial.print(Ax); Serial.print("\t");                 //แสดงค่ามุม Ax
  Serial.print(Ay); Serial.print("\t");                 //แสดงค่ามุม Ay
  Serial.println(Az);                                   //แสดงค่ามุม Az
  delay(100);
}
```

ผลการรันโปรแกรม

โปรแกรมจะอ่านค่า Ax Ay และ Az ซึ่งเป็นข้อมูลดิบ (Raw Value) ที่มีค่าอยู่ในช่วง -18000 ถึง +18000 เป็นการเปลี่ยนแปลงตามตำแหน่งหรือมุม ซึ่งการเปลี่ยนแปลงจะเกิดขึ้นเมื่อโมดูลหรือบอร์ด GY-521 MPU6050 มีการเอียงหรือเคลื่อนที่



รูปที่ 10.4 แสดงข้อมูล Ax Ay และ Az

ตัวอย่างที่ 10.2 โปรแกรมอ่านค่ามุม 0 ถึง 180 ในแนวแกน x y และ z

```
#include <MPU6050.h>                              //เรียกใช้ไลบรารี MPU6050.h
MPU6050 mpu;                                      //ประกาศตัวแปร mpu
int Ax, Ay, Az;                                   //ประกาศตัวแปร Ax Ay Az
int Gx, Gy, Gz;                                   //ประกาศตัวแปร Gx Gy Gz
void setup()
{
  Serial.begin(9600);                             //กำหนดอัตราการสื่อสารแบบอนุกรม
  mpu.initialize();                               //เริ่มการทำงาน MPU6050
}
void loop()
{
  mpu.getMotion6(&Ax,&Ay,&Az,&Gx,&Gy,&Gz);        //อ่านค่ามุมและ Gyro
  Ax = map(Ax, -18000, 18000, 0, 180);            //แปลงค่ามุม Ax (0–180)
  Ay = map(Ay, -18000, 18000, 0, 180);
  Az = map(Az, -18000, 18000, 0, 180);
  Gx = map(Gx, -18000, 18000, 0, 180);            //แปลงค่า Gx (0–180)
  Gy = map(Gy, -18000, 18000, 0, 180);
  Gz = map(Gz, -18000, 18000, 0, 180);
  Serial.print("Axyz \t");                        //แสดงข้อความ Axyz
  Serial.print(Ax); Serial.print("\t");           //แสดงค่ามุม Ax
  Serial.print(Ay); Serial.print("\t");
  Serial.print(Az); Serial.print("\t");
  Serial.print("Gxyz \t");                        //แสดงข้อความ Gxyz
  Serial.print(Gx); Serial.print("\t");           //แสดงค่า Gx
  Serial.print(Gy); Serial.print("\t");
  Serial.println(Gz);
  delay(100);
}
```

ผลการรันโปรแกรม

โปรแกรมจะอ่านค่า Ax Ay Az และ Gx Gy Gz จากเซนเซอร์ซึ่งเป็นข้อมูลดิบ มีค่าอยู่ในช่วง –18000 ถึง +18000 แล้วแปลงเป็นค่ามุมให้อยู่ในช่วง 0–180 องศา โดยที่ Ax Ay Az คือค่ามุมของแต่ละแกน ส่วน Gx Gy Gz คือค่าไจโรซึ่งจะเปลี่ยนแปลงเมื่อมีการเคลื่อนที่ แล้วจะกลับมามีค่าเข้าใกล้ 90 องศา เมื่อหยุดนิ่ง โดยจะไม่เปลี่ยนแปลงตามตำแหน่งหรือมุม แสดงดังรูป

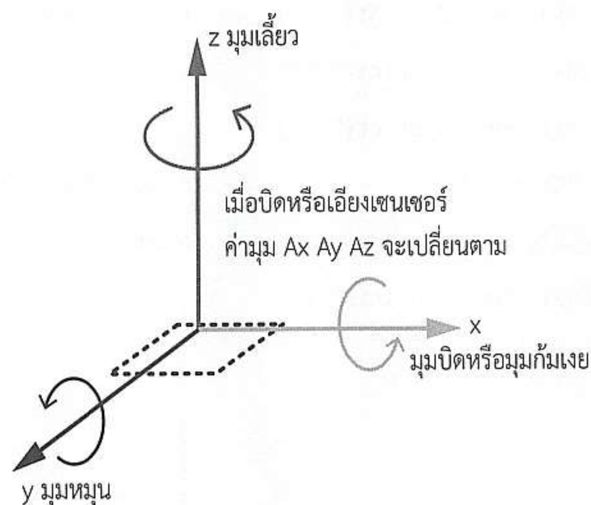
COM11

	Ax	Ay	Az		Gx	Gy	Gz
Axyz	94	78	166	Gxyz	87	90	89
Axyz	94	78	168	Gxyz	87	90	89
Axyz	93	78	167	Gxyz	88	90	89
Axyz	93	79	168	Gxyz	87	90	89
Axyz	93	78	167	Gxyz	87	89	89
Axyz	93	78	167	Gxyz	87	90	89
Axyz	94	78	167	Gxyz	88	90	89
Axyz	93	78	167	Gxyz	88	90	89
Axyz	93	78	168	Gxyz	87	90	89
Axyz	94	78	167	Gxyz	87	89	89
Axyz	94	78	167	Gxyz	88	90	89
Axyz	93	78	168	Gxyz	88	90	89
Axyz	93	78	168	Gxyz	88	89	89
Axyz	93	78	168	Gxyz	88	89	89

ค่ามุมในแนวแกน x y z จะเปลี่ยนแปลงตามมุมเอียงของบอร์ด

Gyro x y z จะเปลี่ยนแปลงเมื่อมีการเคลื่อนที่ แล้วจะกลับมามีค่าเข้าใกล้ 90 องศาเมื่อหยุดนิ่ง

รูปที่ 10.5 แสดงค่ามุม Ax Ay Az และ Gx Gy Gz



รูปที่ 10.6 มุมต่าง ๆ ตามแนวแกน x y และ z ในระบบพิกัดฉาก

ตัวอย่างที่ 10.3 โปรแกรมอ่านค่ามุมโดยไม่ใช้ไลบรารี

```
#include <Wire.h>                               //เรียกใช้ไลบรารี Wire.h
int GyroAddress = 0x68;                         //กำหนด Address Gyro
int GyroPitch, GyroYaw, GyroRoll;
void setup()
{ Serial.begin(9600);
  Wire.begin();
  Wire.beginTransmission(GyroAddress);          //สื่อสารกับ Gyro Address 0x86
  Wire.write(0x6B);                             //เขียนเรจิสเตอร์ 0x6B
  Wire.write(0x00);                             //0x6B = 0 เริ่มการทำงาน
  Wire.endTransmission();                       //จบการสื่อสาร
  Wire.beginTransmission(GyroAddress);
  Wire.write(0x1B);                             //เขียนเรจิสเตอร์ 0x1B
  Wire.write(0x00);                             //0x1B = 0±250 องศาต่อวินาที
  Wire.endTransmission();
  Wire.beginTransmission(GyroAddress);
  Wire.write(0x1C);                             //เขียนเรจิสเตอร์ 0x1C
  Wire.write(0x08);                             //กำหนดให้ Gyro วัดได้ ±4g
  Wire.endTransmission();
}
void loop()
{ Wire.beginTransmission(GyroAddress);          //สื่อสารกับ Gyro Address 0x86
  Wire.write(0x43);                             //ติดต่อเรจิสเตอร์ 0x43
  Wire.endTransmission();
  Wire.requestFrom(GyroAddress,6);              //อ่านค่า Gyro 6 ไบต์จากเรจิสเตอร์ 0x43
  GyroYaw=Wire.read()<<8|Wire.read();           //อ่านค่า 2 ไบต์แล้วนำมารวมกัน
  GyroPitch=Wire.read()<<8|Wire.read();
  GyroRoll=Wire.read()<<8|Wire.read();
  GyroYaw=map(GyroYaw, -18000, 18000, 0, 180);
  GyroPitch=map(GyroPitch, -18000, 18000, 0, 180);
  GyroRoll=map(GyroRoll, -18000, 18000, 0, 180);
  Serial.print("GYaw="); Serial.println(GyroYaw);          //แสดงค่า Gyro Yaw
  Serial.print("GPitch="); Serial.println(GyroPitch);      //แสดงค่า Gyro Pitch
  Serial.print("GRoll="); Serial.println(GyroRoll);        //แสดงค่า Gyro Roll
delay(1000);
}
```

ผลการรันโปรแกรม

โปรแกรมจะอ่านค่าไจโร Yaw Pitch Roll ทั้ง 3 แกนจากเซนเซอร์ซึ่งเป็นข้อมูลดิบ โดยไม่เรียกใช้ไลบรารี แล้วแปลงเป็นค่ามุมที่อยู่ในช่วง 0–180 องศา โดยค่าไจโรทั้ง 3 แกนจะเปลี่ยนแปลงเมื่อมีการเคลื่อนที่แล้วกลับมามีค่าเข้าใกล้ 90 องศาเมื่อหยุดนิ่ง และค่าจะไม่เปลี่ยนแปลงตามมุม แสดงดังรูปที่ 10.7

COM4

Send

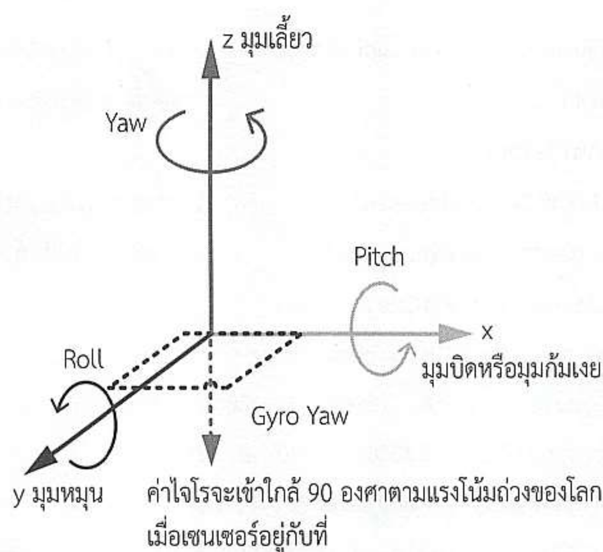
```
GPitch=107
GRoll=127
GYaw=90
GPitch=84
GRoll=100
GYaw=88
GPitch=89
GRoll=89
GYaw=87
GPitch=89
GRoll=89
GYaw=88
GPitch=90
GRoll=89
```

Gyro Sensor จะมีค่าเปลี่ยนไปตามการเคลื่อนไหว

เมื่อ Gyro Sensor อยู่นิ่งกับที่จะมีค่าเข้าใกล้ 90 องศา

Autoscroll   Both NL & CR   9600 baud

รูปที่ 10.7 แสดงค่าไจโร Yaw Pitch Roll



รูปที่ 10.8 ระบบพิกัดฉากของไจโร

10.3 วงจรควบคุมหุ่นยนต์ 2 ล้อสมดุล

Pin 2 4 -> STEP ขาสัญญาณพัลส์ควบคุมความเร็วและตำแหน่ง

Pin 3 5 -> DIR ขาสัญญาณควบคุมทิศทาง

รูปที่ 10.9 วงจรควบคุมหุ่นยนต์ 2 ล้อสมดุล

จากรูปที่ 10.9 วงจรควบคุมหุ่นยนต์ 2 ล้อสมดุลประกอบด้วยอุปกรณ์ต่าง ๆ ดังนี้

1) ไมโครคอนโทรลเลอร์ Arduino ทำหน้าที่ควบคุมสเต็ปเปอร์มอเตอร์ให้เดินหน้าหรือถอยหลังตามการเอียงหรือมุมก้มเงยของหุ่นยนต์ โดยใช้ระบบควบคุมแบบ PID เพื่อคำนวณหาค่าทิศทางและความเร็วในการหมุนของสเต็ปเปอร์มอเตอร์

2) GY-521 MPU6050 เป็นเซนเซอร์วัดมุมเอียงของหุ่นยนต์

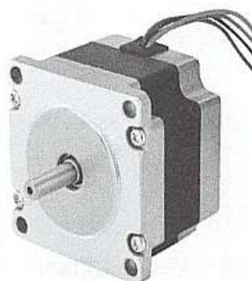
3) DRV8825 ทำหน้าที่ขับสเต็ปเปอร์มอเตอร์ให้หมุนไปตามตำแหน่งและทิศทางที่ต้องการ ใช้แรงดันไฟฟ้าจากแบตเตอรี่ 11.1 โวลต์ ขับมอเตอร์ (หากจ่ายแรงดัน VMOT น้อยกว่า 8 โวลต์ วงจรจะไม่ทำงาน) ใช้ไฟเลี้ยงวงจร 3.3–5 โวลต์ ขับกระแสไฟฟ้าสูงสุดได้ 2.5 แอมแปร์ สามารถปรับความละเอียดมุมในการหมุนของมอเตอร์ได้โดยการปรับโหมดการทำงาน M0 M1 และ M2

4) แบตเตอรี่ลิเทียมพอลิเมอร์ (Lithium Polymer : Li-Po) 3 เซลล์ 11.1 โวลต์ ต่อเข้ากับขั้วจ่ายไฟ (Power Jack) ของไมโครคอนโทรลเลอร์ และต่อเข้ากับขา VMOT และ GND ของบอร์ด DRV8825

5) สเต็ปเปอร์มอเตอร์ทำหน้าที่หมุนซ้ายหรือหมุนขวา เพื่อควบคุมหุ่นยนต์ให้เดินหน้าหรือถอยหลังสามารถควบคุมมุม ตำแหน่ง และจำนวนรอบในการหมุนได้ เป็นสเต็ปเปอร์มอเตอร์แบบไบโพลาร์ 2 เฟส 4 สาย ขนาด 35 มิลลิเมตร 1.5 แอมแปร์ โดยมีคุณสมบัติดังนี้

คุณลักษณะของสเต็ปเปอร์มอเตอร์

- Motor Height: 23.5 mm
- Output Shaft: 3 mm
- Output Shaft Length: 15 mm
- Step Angle: 1.8°
- Lead: 2 Phase 4 Wire
- Phase Voltage: 2.4 V
- Driving Current: 1.5 A
- Phase Resistance: 1.7 Ohm
- Torque: 0.08 Nm
- Weight: 124 g



รูปที่ 10.10 สเต็ปเปอร์มอเตอร์ขนาด 35 มิลลิเมตร

ตัวอย่างที่ 10.4 โปรแกรมทดสอบสเต็ปเปอร์มอเตอร์

```
int Lmotor,SpeedLmotor,countLmotor,Lmotor_mem;
int Rmotor,SpeedRmotor,countRmotor,Rmotor_mem;
int c=0;
double TM;
void setup()
{
  TCCR2A = 0;                    //ให้เรจิสเตอร์ TCCR2A = 0
  TCCR2B = 0;                    //ให้เรจิสเตอร์ TCCR2B = 0
  TIMSK2 |= (1 << OCIE2A);       //เลื่อนบิตแล้ว OR เก็บไว้ในเรจิสเตอร์ TIMSK2
  TCCR2B |= (1 << CS21);         //เลื่อนบิตแล้ว OR เก็บไว้ในเรจิสเตอร์ TCCR2B
  OCR2A = 39;                    //ให้เรจิสเตอร์ OCR2A = 39
  TCCR2A |= (1 << WGM21);        //เลื่อนบิตแล้ว OR เก็บไว้ในเรจิสเตอร์ TCCR2A
  pinMode(2, OUTPUT);            //Step Left Motor
  pinMode(3, OUTPUT);            //DIR Left Motor
  pinMode(4, OUTPUT);            //Step Right Motor
  pinMode(5, OUTPUT);            //DIR Right Motor
}
void loop()
{
  if(TM<millis())                //ถ้า TM < millis()
  {
    TM=millis()+5000;            //บวกค่าตัวแปร TM
    c=!c;                        //กลับบิตตัวแปร c ทุก ๆ 5000 ms
  }
  if(c==1)                       //ถ้า c เท่ากับ 1
  {
    Lmotor=-100;                 //ให้ Lmotor = –100 ให้หมุนทวนเข็ม (ถอยหลัง)
    Rmotor=-100;                 //ให้ Rmotor = –100 ให้หมุนทวนเข็ม (ถอยหลัง)
  }
  else
```

```
   {
     Lmotor=0;                              //ให้ Lmotor = 0 มอเตอร์ด้านซ้ายหยุดหมุน
     Rmotor=0;                              //ให้ Rmotor = 0 มอเตอร์ด้านขวาหยุดหมุน
   }
   SpeedLmotor = Lmotor;                    //ให้ SpeedLmotor = Lmotor
   SpeedRmotor = Rmotor;                    //ควบคุมความเร็ว (100) และทิศทาง (+/–)
 }
 //Interrupt routine  TIMER2_COMPA_vect
 ISR(TIMER2_COMPA_vect)                     //อินเทอร์รัปต์ Timer
 {
   //Left motor pulse calculations
   countLmotor ++;                          //นับเพื่อสร้างสัญญาณพัลส์
   if (countLmotor > Lmotor_mem)            //ถ้ามากกว่า
   {
   countLmotor = 0;
   Lmotor_mem = SpeedLmotor;                //Lmotor_mem = ความเร็วของมอเตอร์
   if(Lmotor_mem < 0)                       //ถ้ามีค่าเป็นลบ
     {
         PORTD &= 0b11110111;               //ทิศทางการหมุน ขา 3 DIR Lmotor = 0
         Lmotor_mem = Lmotor_mem*-1;        //ให้ความเร็วมีค่าเป็นบวก
     }
   else PORTD |= 0b00001000;                //ทิศทางการหมุน ขา 3 DIR Lmotor = 1
   }
 else if (countLmotor == 1)PORTD |= 0b00000100; //สร้างสัญญาณพัลส์ 1
 else if (countLmotor == 2)PORTD &= 0b11111011; //สร้างสัญญาณพัลส์ 0
 //right motor pulse calculations
 countRmotor ++;                            //นับเพื่อสร้างสัญญาณพัลส์
 if (countRmotor > Rmotor_mem)              //ถ้ามากกว่า
   {
     countRmotor = 0;
     Rmotor_mem = SpeedRmotor;              //Rmotor_mem = ความเร็วของมอเตอร์
     if (Rmotor_mem < 0)                    //ถ้ามีค่าเป็นลบ
```

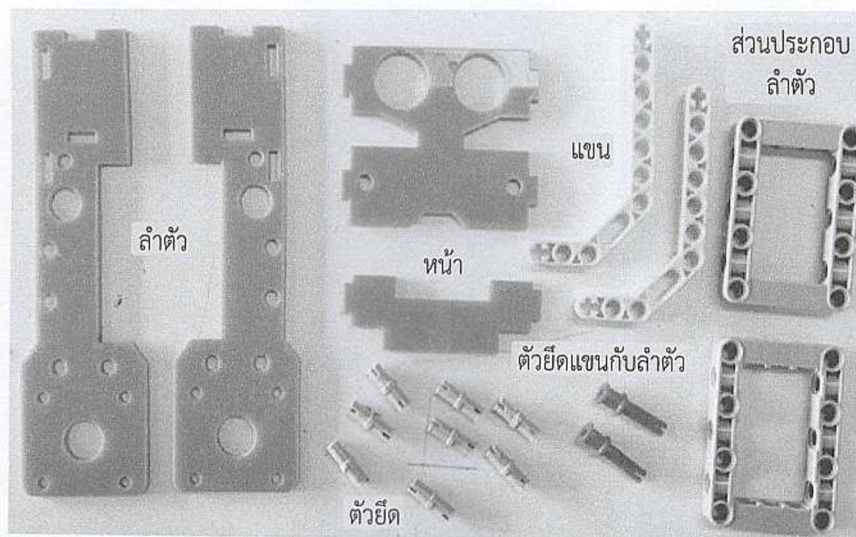
```
    {
      PORTD |= 0b00100000;                  //ทิศทางการหมุน ขา 5 DIR = 1
      Rmotor_mem = Rmotor_mem*-1;           //ให้ความเร็วมีค่าเป็นบวก
    }
  else PORTD &= 0b11011111;                 //ทิศทางการหมุน ขา 5 DIR = 0
  }
else if (countRmotor == 1)PORTD |= 0b00010000; //สร้างสัญญาณพัลส์ 1
else if (countRmotor == 2)PORTD &= 0b11101111; //สร้างสัญญาณพัลส์ 0
}
```

ผลการรันโปรแกรม

สเต็ปเปอร์มอเตอร์จะหมุนและหยุดหมุนตามเงื่อนไข คือถ้าตัวแปร c = 1 มอเตอร์จะหมุนด้วยความเร็ว (100) ตามความเร็วที่กำหนด ส่วนทิศทางการหมุนจะขึ้นอยู่กับค่าบวกหรือลบ ถ้า c = 0 มอเตอร์จะหยุดหมุน โดยการเขียนโปรแกรมแบบอินเทอร์รัปต์เพื่อไม่ต้องใช้ฟังก์ชัน delay();

10.4 การสร้างหุ่นยนต์ 2 ล้อสมดุล

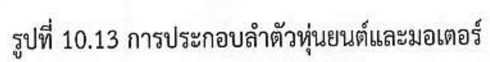
โครงสร้างของหุ่นยนต์ 2 ล้อสมดุลอย่างง่ายในที่นี้จะใช้ชิ้นส่วนตัวต่อ ล้อพลาสติกสำเร็จรูป และพลาสติกที่ตัดด้วยเครื่องตัดเลเซอร์ โดยมีขั้นตอนการประกอบดังรูปที่ 10.11–10.15



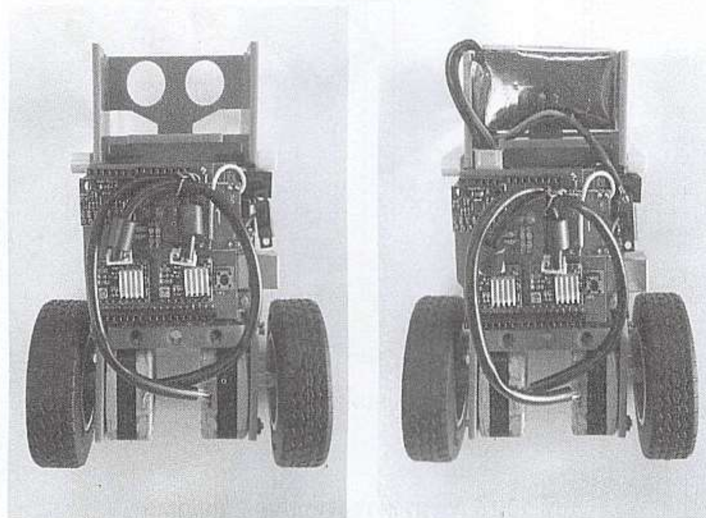
รูปที่ 10.11 ชิ้นส่วนโครงสร้างหุ่นยนต์ 2 ล้อสมดุล



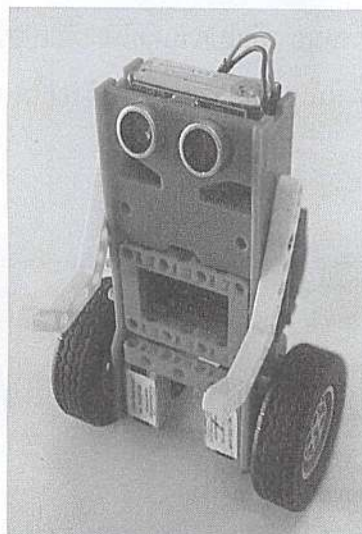
รูปที่ 10.12 แบบพลาสติกสำหรับทำตัวหุ่นยนต์



รูปที่ 10.13 การประกอบลำตัวหุ่นยนต์และมอเตอร์



รูปที่ 10.14 การประกอบวงจรและแบตเตอรี่

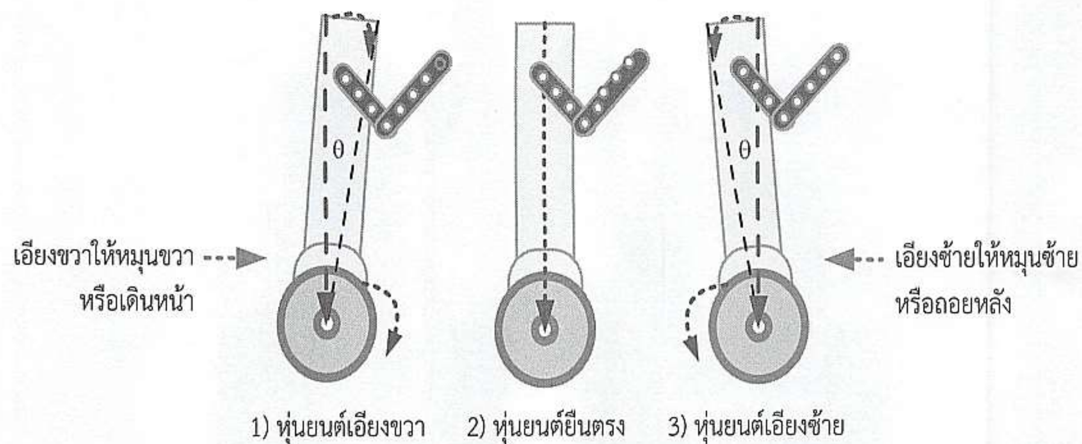


รูปที่ 10.15 หุ่นยนต์ 2 ล้อสมดุลที่ประกอบเสร็จ

10.5 หลักการทรงตัวของหุ่นยนต์ 2 ล้อสมดุล

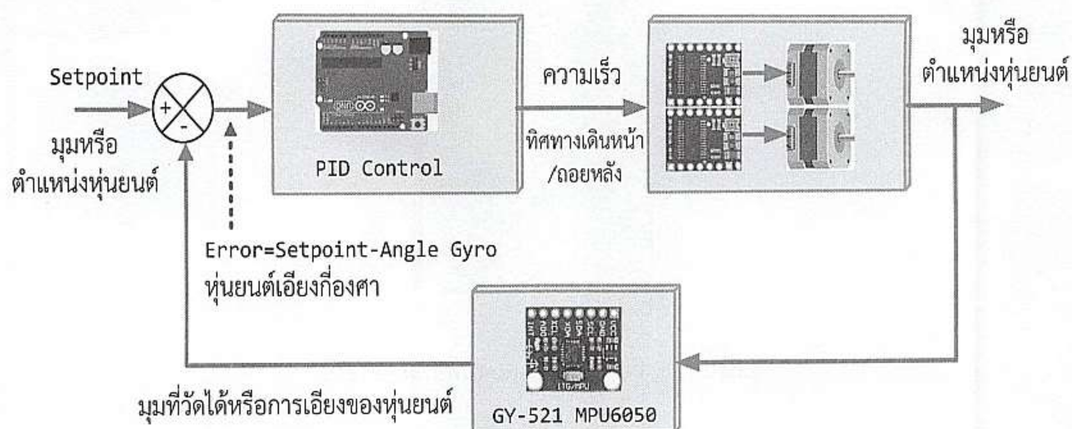
หลักการควบคุมหุ่นยนต์ 2 ล้อสมดุลให้ทรงตัวอยู่ได้ จะใช้เซนเซอร์วัดมุมเอียงของหุ่นยนต์แล้วส่งสัญญาณค่าผิดพลาดให้ไมโครคอนโทรลเลอร์ควบคุมแบบ PID เพื่อไปบังคับสเต็ปเปอร์มอเตอร์ทั้งสองตัวให้ทำงานตามเงื่อนไขดังนี้

1) ถ้าหุ่นยนต์เอียงขวาหรือก้มหน้า ให้หุ่นยนต์หมุนไปทางขวาหรือเดินหน้า
2) ถ้าหุ่นยนต์ยืนตรงอยู่ในสมดุล ให้มอเตอร์หยุดหมุนหรือหมุนเล็กน้อย
3) ถ้าหุ่นยนต์เอียงซ้ายหรือเงยหน้า ให้หุ่นยนต์หมุนไปทางซ้ายหรือถอยหลัง



รูปที่ 10.16 การทรงตัวของหุ่นยนต์ 2 ล้อสมดุล

ส่วนการควบคุมให้มอเตอร์เดินหน้าหรือถอยหลัง ไมโครคอนโทรลเลอร์จะนำค่ามุมเอียงหรือมุมที่ผิดพลาด (Error = Setpoint – Angle Gyro) มาผ่านสมการ PID หรือการควบคุมแบบ PID ได้ผลลัพธ์ออกมาเป็นทิศทางและความเร็วในการหมุน เพื่อไปควบคุมให้สเต็ปเปอร์มอเตอร์เดินหน้าหรือถอยหลังด้วยความเร็วที่ปรับเปลี่ยนตามการเอียงทำให้หุ่นยนต์สามารถทรงตัวอยู่ได้ แสดงบล็อกไดอะแกรมการควบคุมหุ่นยนต์ดังรูปที่ 10.17



รูปที่ 10.17 บล็อกไดอะแกรมการควบคุมหุ่นยนต์ 2 ล้อสมดุล

ตัวอย่างที่ 10.5 โปรแกรมควบคุมการทรงตัวของหุ่นยนต์ 2 ล้อสมดุล

```
#include <Wire.h>                                //เรียกใช้ไลบรารี Wire.h
int GyroAddress = 0x68;                          //กำหนด Gyro Address = 0x68
int Acc_calibration = 800;                       //ประกาศตัวแปรและปรับค่า Gyro
int start, x, counter;
int Lmotor, SpeedLmotor, countLmotor, Lmotor_mem;
int Rmotor, SpeedRmotor, countRmotor, Rmotor_mem;
int GyroPitch, GyroYaw, Accelerometer;
long GyroYaw_calibration, GyroPitch_calibration;
unsigned long loop_timer;
float Kp = 15;
float Ki = 1.5;
float Kd = 30;
float AngleGyro, AngleAcc, Angle, BalanceAdj;
float Error, P, I, D, Setpoint, PIDoutput, PreError;
float outputLeft, outputRight;
void setup()
{
   Serial.begin(9600);                           //กำหนดการสื่อสารแบบอนุกรม
   Wire.begin();                                 //เริ่มการสื่อสารแบบ I2C
   //กำหนดค่าเรจิสเตอร์เกี่ยวกับ Interrupt Timer
   TWBR = 12;
   TCCR2A = 0;
   TCCR2B = 0;
   TIMSK2 |= (1 << OCIE2A);
   TCCR2B |= (1 << CS21);
   OCR2A = 39;
   TCCR2A |= (1 << WGM21);
   Wire.beginTransmission(GyroAddress);          //สื่อสารกับ Gyro Address 0x86
   Wire.write(0x6B);                             //เขียนเรจิสเตอร์ 0x6B
   Wire.write(0x00);                             //0x6B = 0 เริ่มการทำงาน Gyro
   Wire.endTransmission();                       //จบการสื่อสาร
   Wire.beginTransmission(GyroAddress);
```

```
  Wire.write(0x1B);                        //เขียนเรจิสเตอร์ 0x1B
  Wire.write(0x00);                        //0x1B = ±250 องศาต่อวินาที
  Wire.endTransmission();
  Wire.beginTransmission(GyroAddress);
  Wire.write(0x1C);                        //เขียนเรจิสเตอร์ 0x1C
  Wire.write(0x08);                        //กำหนดให้ Gyro วัดได้ ±4g
  Wire.endTransmission();
  Wire.beginTransmission(GyroAddress);
  Wire.write(0x1A);                        //เขียนเรจิสเตอร์ 0x1A
  Wire.write(0x03);                        //0x1A = 03
  Wire.endTransmission();
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode (5, OUTPUT);
  pinMode(13, OUTPUT);
  //วนรอบ 500 ครั้ง บวกค่า GyroYaw GyroPitch และให้ไฟกะพริบที่ขา 13
  for (counter = 0; counter < 500; counter++)
  {
    if (counter % 20 == 0)
    digitalWrite(13, !digitalRead(13));
    Wire.beginTransmission(GyroAddress);
    Wire.write(0x43);
    Wire.endTransmission();
    Wire.requestFrom(GyroAddress, 4);
    GyroYaw_calibration += Wire.read() << 8 | Wire.read();
    GyroPitch_calibration += Wire.read() << 8 | Wire.read();
    delayMicroseconds(3700);
  }
//หาค่าเฉลี่ยของ GyroYaw GyroPitch หารด้วย 500
GyroYaw_calibration = GyroYaw_calibration / 500;
GyroPitch_calibration = GyroPitch_calibration / 500;
loop_timer = micros() + 4000;
```

```
}
void loop()
{
//อ่านค่าความเร่งจาก Gyro เพื่อคำนวณหามุมเอียง
Wire.beginTransmission(GyroAddress);
Wire.write(0x3F);
Wire.endTransmission();
Wire.requestFrom(GyroAddress, 2);
Accelerometer = Wire.read() << 8 | Wire.read();
Accelerometer = Accelerometer+Acc_calibration;
if (Accelerometer > 8200) Accelerometer = 8200;
if (Accelerometer < -8200) Accelerometer = -8200;
//คำนวณหามุมเอียงหรือมุมก้มเงยของหุ่นยนต์
AngleAcc = asin((float)Accelerometer/8200.0) *57.296;
//ถ้า start = 0 และมุมเอียงอยู่ในช่วง –0.5 ถึง +0.5 หรือหุ่นยนต์เอียงเล็กน้อย
if (start == 0 && AngleAcc > -0.5 && AngleAcc < 0.5)
  {
     AngleGyro = AngleAcc;
     start = 1;
  }
//อ่านค่ามุมเอียงและมุมเลี้ยว GyroPitch GyroYaw
Wire.beginTransmission(GyroAddress);
Wire.write(0x43);
Wire.endTransmission();
Wire.requestFrom(GyroAddress, 4);
GyroYaw = Wire.read() << 8 | Wire.read();
GyroPitch = Wire.read() << 8 | Wire.read();
GyroPitch -= GyroPitch_calibration;
AngleGyro += GyroPitch * 0.000031;
GyroYaw -= GyroYaw_calibration;
AngleGyro = AngleGyro * 0.9996 + AngleAcc * 0.0004;
//คำนวณหาค่าผิดพลาดหรือมุมเอียงของหุ่นยนต์
Error = AngleGyro - Setpoint - BalanceAdj;
```

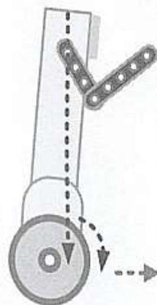
```
if (PIDoutput > 10 || PIDoutput < -10)
   Error = Error + PIDoutput * 0.015 ;
//คำนวณหาค่า PID
P = Kp * Error;
I = (Ki * Error) + I;
D = Kd * (Error - PreError);
PIDoutput = P + I + D;
if (PIDoutput > 400) PIDoutput = 400;
if (PIDoutput < -400) PIDoutput = -400;
PreError = Error;                         //เก็บค่าผิดพลาด PreError
if (PIDoutput < 5 && PIDoutput > -5)
   PIDoutput = 0;
//ถ้าหุ่นยนต์เอียงเกิน 30 หรือ start = 0 ให้รีเซ็ตค่าตัวแปร
if (AngleGyro > 30 || AngleGyro < -30 || start == 0)
   {
      PIDoutput = 0;
      I = 0;
      start = 0;
      BalanceAdj = 0;
   }
//Control calculations
outputLeft = PIDoutput;
outputRight = PIDoutput;
//ปรับค่า Setpoint
if (Setpoint > 0.5)
   Setpoint = Setpoint - 0.05;
else if (Setpoint < -0.5)
   Setpoint = Setpoint + 0.05;
else
   Setpoint = 0;
//หุ่นยนต์เอียงเล็กน้อยให้ปรับละเอียด
if (Setpoint == 0)
   {
```

```
   if (PIDoutput < 0)
      BalanceAdj = BalanceAdj + 0.0015;
   if (PIDoutput > 0)
      BalanceAdj = BalanceAdj - 0.0015;
   }
//คำนวณหาค่าความเร็วในการหมุนสเต็ปเปอร์มอเตอร์
if (outputLeft > 0)
   outputLeft = 405 - (1 / (outputLeft + 9)) * 5500;
else if (outputLeft < 0)
   outputLeft = -405 - (1 / (outputLeft - 9)) * 5500;
if (outputRight > 0)
   outputRight = 405 - (1 / (outputRight + 9)) * 5500;
else if (outputRight < 0)
   outputRight = -405 - (1 / (outputRight - 9)) * 5500;
//คำนวณหาค่าความเร็วมอเตอร์ด้านซ้าย
if (outputLeft > 0)                          //ถ้ามีค่ามากกว่าศูนย์
   Lmotor = 400 - outputLeft;
else if (outputLeft < 0)                     //ถ้ามีค่าน้อยกว่าศูนย์
  Lmotor = -400 - outputLeft;
else                                         //ถ้ามีค่าเท่ากับศูนย์
  Lmotor = 0;
//คำนวณหาค่าความเร็วมอเตอร์ด้านขวา
if (outputRight > 0)
   Rmotor = 400 - outputRight;
else if (outputRight < 0)
   Rmotor = -400 - outputRight;
else Rmotor = 0;
SpeedLmotor = Lmotor;
SpeedRmotor = Rmotor;
//วงรอบการทำงานเท่ากับ 4000 ไมโครวินาที
while (loop_timer > micros());
loop_timer += 4000;
}
```

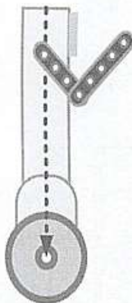
```
//โปรแกรมบริการอินเทอร์รัปต์ควบคุมสเต็ปเปอร์มอเตอร์ให้หมุนตามเวลาหรือความเร็วที่ส่งมา
ISR(TIMER2_COMPA_vect)                    //อินเทอร์รัปต์ Timer
{
   //Left motor pulse calculations
   countLmotor ++;                        //นับเพื่อสร้างสัญญาณพัลส์
   if (countLmotor > Lmotor_mem)          //ถ้ามากกว่า
   {
      countLmotor = 0;
      Lmotor_mem = SpeedLmotor;           //Lmotor_mem = ความเร็วของมอเตอร์
      if (Lmotor_mem < 0)                 //ถ้ามีค่าเป็นลบ
      {
         PORTD &= 0b11110111;             //ทิศทางการหมุน ขา 3 DIR = 0
         Lmotor_mem = Lmotor_mem * -1;    //ให้ความเร็วมีค่าเป็นบวก
      }
   else PORTD |= 0b00001000;              //ทิศทางการหมุน ขา 3 DIR = 1
   }
   else if (countLmotor == 1)PORTD |= 0b00000100;    //สร้างสัญญาณพัลส์ 1
   else if (countLmotor == 2)PORTD &= 0b11111011;    //สร้างสัญญาณพัลส์ 0
   //right motor pulse calculations
   countRmotor ++;                        //นับเพื่อสร้างสัญญาณพัลส์
   if (countRmotor > Rmotor_mem)          //ถ้ามากกว่า
   {
      countRmotor = 0;
      Rmotor_mem = SpeedRmotor;           //Rmotor_mem = ความเร็วของมอเตอร์
      if (Rmotor_mem < 0)                 //ถ้ามีค่าเป็นลบ
      {
         PORTD |= 0b00100000;             //ทิศทางการหมุน ขา 5 DIR = 1
         Rmotor_mem = Rmotor_mem * -1;    //ให้ความเร็วมีค่าเป็นบวก
      }
      else PORTD &= 0b11011111;           //ทิศทางการหมุน ขา 5 DIR = 0
   }
   else if (countRmotor == 1)PORTD |= 0b00010000;    //สร้างสัญญาณพัลส์ 1
   else if (countRmotor == 2)PORTD &= 0b11101111;    //สร้างสัญญาณพัลส์ 0
}
```

ผลการรันโปรแกรม

หุ่นยนต์จะพยายามทรงตัวโดยควบคุมการเคลื่อนที่ให้เดินหน้าหรือถอยหลังด้วยความเร็วในการหมุนที่ต่างกันตามการเอียงของหุ่นยนต์ โดยโปรแกรมจะคำนวณหาค่าผิดพลาดหรือมุมเอียง จากนั้นนำค่ามุมที่ได้มาผ่านการควบคุมแบบ PID แล้วคำนวณเป็นทิศทางและความเร็วในการหมุนของสเต็ปเปอร์มอเตอร์ดังรูปที่ 10.18

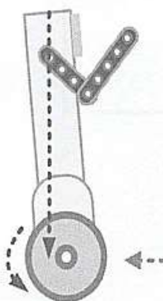


ถ้าหุ่นยนต์เอียงขวาหรือเอียงไปด้านหน้า ให้หุ่นยนต์เดินหน้าด้วยความเร็วที่ต่างกันตามการเอียงของหุ่นยนต์ ถ้าหุ่นยนต์เอียงมากให้หมุนเร็วขึ้น



```
//หุ่นยนต์เอียงเล็กน้อยให้ปรับละเอียด
if (Setpoint == 0)
  { if (PIDoutput < 0)
    BalanceAdj = BalanceAdj + 0.0015;
    if (PIDoutput > 0)
    BalanceAdj = BalanceAdj - 0.0015;
  }
```

ถ้าหุ่นยนต์ยืนตรงให้มอเตอร์หยุดหมุนหรือหมุนช้า ๆ เพื่อรักษาสมดุล



ถ้าหุ่นยนต์เอียงซ้ายหรือเอียงไปด้านหลัง ให้หุ่นยนต์ถอยหลังด้วยความเร็วที่ต่างกันตามการเอียงของหุ่นยนต์ ถ้าหุ่นยนต์เอียงมากให้หมุนเร็วขึ้น

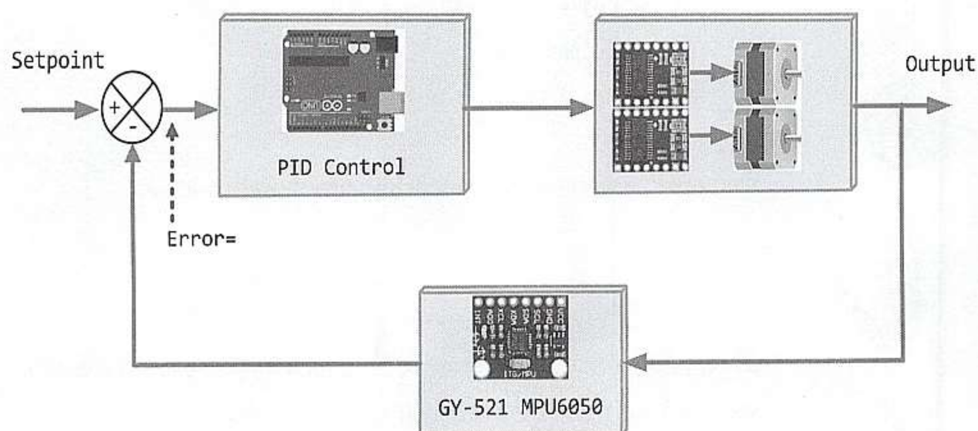
รูปที่ 10.18 การควบคุมการทรงตัวของหุ่นยนต์

10.6 สรุป

หุ่นยนต์ 2 ล้อสมดุล คือหุ่นยนต์ที่ใช้เพียง 2 ล้อในการเคลื่อนที่และยังคงทรงตัวอยู่ได้โดยไม่ล้ม หุ่นยนต์ชนิดนี้จะใช้เซนเซอร์วัดมุมเพื่อหาค่ามุมเอียงหรือค่าผิดพลาด แล้วคำนวณตามสมการหรือการควบคุมแบบ PID เพื่อหาทิศทางและความเร็วในการหมุนของสเต็ปเปอร์มอเตอร์เพื่อปรับสมดุลให้หุ่นยนต์ทรงตัวได้

แบบฝึกหัดบทที่ 10

1. อธิบายหลักการทำงานของโมดูล GY-521 MPU6050
2. อธิบายหลักการทรงตัวของหุ่นยนต์
3. เขียนโปรแกรมควบคุมสเต็ปเปอร์มอเตอร์ให้เดินหน้าแล้วถอยหลังครั้งละ 1 นาที
4. อธิบายฟังก์ชันหรือคำสั่งต่าง ๆ ดังนี้
   - Wire.beginTransmission(GyroAddress);
   - Wire.write(0x43);
   - Wire.endTransmission();
   - Wire.requestFrom(GyroAddress, 4);
   - GyroYaw = Wire.read() << 8 | Wire.read();
   - GyroPitch = Wire.read() << 8 | Wire.read();
5. อธิบายการทำงานของบล็อกไดอะแกรมการควบคุมหุ่นยนต์ดังรูปที่ 10.19

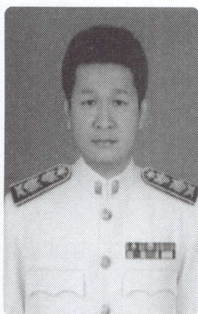


รูปที่ 10.19 บล็อกไดอะแกรมการควบคุมหุ่นยนต์ 2 ล้อสมดุล

บรรณานุกรม

1. ดอนสัน ปงผาบ. **ภาษาซีและ Arduino.** กรุงเทพฯ : สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2560.
2. Banzi, Massimo. **Getting Started with Arduino.** USA: O'Reilly Media, 2011.
3. Purdum, Jack. **Beginning C for Arduino.** USA: Apress, 2012.
4. Monk, Simon. **Programming Arduino.** McGraw-Hill Education TAB; 2 Edition, 2016.
5. [ระบบออนไลน์]. แหล่งที่มา http://www.arduino.cc\en\Reference\HomePage.html (24 มกราคม 2562).
6. [ระบบออนไลน์]. แหล่งที่มา http://arduinolearning.com/code/arduino-bh1750-sensor.php (12 กุมภาพันธ์ 2562).
7. [ระบบออนไลน์]. แหล่งที่มา http://www.brokking.net/yabr_main.html?fbclid=IwAR0YnHtH6YUyAjXtJTa1knj0deZEVnh0TKTrrD7XzPTXE6zReumPzwEQc0c
8. [ระบบออนไลน์]. แหล่งที่มา https://github.com/CheapskateProjects/WalkingArduinoRobot (1 สิงหาคม 2562).
9. [ระบบออนไลน์]. แหล่งที่มา https://www.instructables.com/id/How-to-Interface-HX711-Balance-Module-With-Load-Ce/ (16 ธันวาคม 2561).
10. [ระบบออนไลน์]. แหล่งที่มา https://th.wikipedia.org/wiki/%E0%B9%84%E0%B8%88%E0%B9%82%E0%B8%A3%E0%B8%AA%E0%B9%82%E0%B8%84%E0%B8%9B (1 มกราคม 2562).

ประวัติผู้เขียน



ผู้ช่วยศาสตราจารย์ดอนสัน ปงผาบ
คณะเทคโนโลยีอุตสาหกรรม มหาวิทยาลัยราชภัฏลำปาง

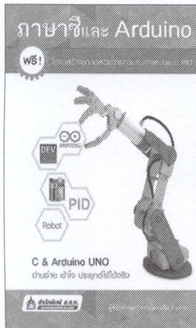
ประวัติการศึกษา

- คอ.บ. อิเล็กทรอนิกส์และคอมพิวเตอร์ (เกียรตินิยม)
  สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
- วศ.ม. วิศวกรรมไฟฟ้า สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ

ช่องทางติดต่อ

- Facebook : Adon_pleanplus
- Line ID : Adonson
- E-mail : donson@lpru.ac.th
- โทร. : 084-5000-474





ภาษาซีและ Arduino

โดย ผู้ช่วยศาสตราจารย์ดอนสัน ปงผาบ

216 หน้า / ราคา 295 บาท

เนื้อหาประกอบด้วย • ความรู้พื้นฐานของภาษาซี • การเขียนโปรแกรม DEV-C++ • ไมโคร-คอนโทรลเลอร์ Arduino UNO • ดิจิทัลและแอนะล็อกอินพุตเอาต์พุต • การควบคุมดีซีมอเตอร์ สเต็ปเปอร์มอเตอร์ เซอร์โวมอเตอร์ • ระบบควบคุมแบบ PID • หุ่นยนต์และการประยุกต์ใช้งานแขนกลขั้นพื้นฐาน นอกจากนี้ยังมีโครงสร้างชุดทดลองการควบคุมตำแหน่งแบบ PID เพื่อใช้ประกอบการศึกษาด้วย



เครื่องกลไฟฟ้า 1 (ฉบับปรับปรุง)

โดย ไชยชาญ หินเกิด

276 หน้า / ราคา 230 บาท

ปรับปรุงจาก “เครื่องกลไฟฟ้า 1” หนังสือที่ได้รับรางวัลหนังสือยอดนิยมของ ส.ส.ท. โดยเพิ่มคำอธิบายและภาพประกอบ ปรับลำดับหัวข้อให้เข้าใจง่ายขึ้น ให้ความรู้พื้นฐานที่สำคัญสำหรับผู้เรียนในสาขาช่างไฟฟ้า เพื่อเป็นแนวทางสู่การศึกษาในระดับสูงขึ้น และเป็นพื้นฐานในการประกอบอาชีพ

เนื้อหาประกอบด้วย • ชนิดและโครงสร้างของเครื่องกลไฟฟ้ากระแสตรง • เครื่องกำเนิดไฟฟ้ากระแสตรง • มอเตอร์ไฟฟ้ากระแสตรง • การเริ่มเดินมอเตอร์ไฟฟ้ากระแสตรง • หม้อแปลงไฟฟ้าเฟสเดียว • การต่อหม้อแปลงไฟฟ้า 3 เฟส





การออกแบบเครื่องจักรกล

โดย ชิเงรุ อิเคดะ, ยูจิ นากานิชิ

264 หน้า / ราคา 260 บาท

กล่าวถึงการออกแบบชิ้นส่วนเครื่องจักรกลสำหรับผู้เริ่มต้น เนื้อหากระชับ เรียนรู้ง่าย อธิบายด้วยภาพประกอบจำนวนมาก มีโจทย์ตัวอย่างการใช้งานจริง และแบบฝึกหัดท้ายบทพร้อมเฉลยอย่างละเอียด

เนื้อหาประกอบด้วย • พื้นฐานการออกแบบเครื่องจักรกล • ชิ้นส่วนการยึด • ชิ้นส่วนระบบเพลา • ตลับลูกปืน • เฟือง • ชิ้นส่วนส่งกำลังแบบม้วนพัน • ชิ้นส่วนกันสะเทือน เหมาะสำหรับนักศึกษาด้านวิศวกรรมเครื่องกล ในระดับ ปวช. ปวส. ปริญญาตรี รวมถึงช่างเทคนิคที่เกี่ยวข้อง



มอเตอร์ไฟฟ้าและการควบคุม

โดย ไชยชาญ หินเกิด

340 หน้า / ราคา 290 บาท

เนื้อหาประกอบด้วย • มอเตอร์ไฟฟ้ากระแสตรง • สปลิตเฟสมอเตอร์ • คาปาซิเตอร์มอเตอร์ • รีพัลชันมอเตอร์ • ยูนิเวอร์แซลมอเตอร์ • เชดเดดโพลมอเตอร์ • ซิงโครนัสมอเตอร์ • มอเตอร์สามเฟส • สัญลักษณ์และอุปกรณ์ที่ใช้ในการควบคุม • ขนาดสายไฟฟ้าและอุปกรณ์ป้องกันมอเตอร์ • การควบคุมมอเตอร์ไฟฟ้ากระแสตรง • การควบคุมมอเตอร์ไฟฟ้ากระแสสลับ เป็นความรู้พื้นฐานของการเรียนในสาขาช่างไฟฟ้า เหมาะสำหรับ นักเรียน นักศึกษา ในระดับ ปวช. และ ปวส. และสามารถใช้เป็นพื้นฐานในการประกอบอาชีพ

ไมโครคอนโทรลเลอร์
Arduino

เนื้อหาประกอบด้วย

พื้นฐานของไมโครคอนโทรลเลอร์ Arduino การเขียนโปรแกรม อุปกรณ์อินพุตและเอาต์พุต

เซนเซอร์ตรวจจับแสง แรงดัน ระยะทาง อุณหภูมิ ความชื้น น้ำหนัก Gyro และเซนเซอร์ตรวจจับโลหะ

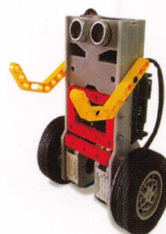
จอแสดงผล LCD OLED LED และ 7-Segment

สเต็ปเปอร์มอเตอร์ เซอร์โวมอเตอร์ ดีซีมอเตอร์ และเอนโค้ดเดอร์มอเตอร์

PID Control

การควบคุมแบบป้อนกลับและการควบคุมแบบ PID

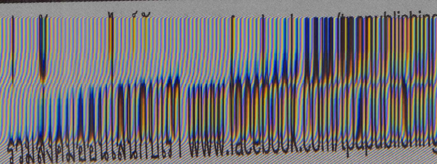
หุ่นยนต์เดินตามเส้น หุ่นยนต์ 4 ขา และหุ่นยนต์ 2 ล้อสมดุล





พบหนังสือออกใหม่ • เสนอผลงานเขียน/แปล ได้ที่

www.tpa.or.th/tpapublishing





ISBN 978-974-443-779-2





หมวดคอมพิวเตอร์